UNIVERSITETET I AGDER

Load-balancing by applying a Bayesian Learning Automata (BLA) scheme in a non-stationary web-crawler network

By

Tarjei Romtveit

Thesis submitted in Partial Fulfillment of the
Requirements for the Degree Master of Science in
Information and Communication Technology

Faculty of Engineering and Science
The University of Agder

Grimstad
May 25, 2010

Abstract

Distributed Web-Crawlers, i.e. , Web-Crawler Networks, have been known to retrieve massive amount of web-data to centralized search indexes the last decade. Companies like Google, Yahoo and lately Integrasco, have built their business model upon this retrieval. However, to load-balance Web-Crawler Networks have been proved difficult. Especially difficult are the geographical distributed Web-Crawler Networks that newly have emerged. In geographically distributed Web-Crawler Networks, the load-balancing algorithm need to consider that the capacity is constantly changing (non-stationary), and location-aware retrieval. The novel approach in this thesis, is to apply the machine-learning technique BLA-Kalman to dynamically load-balance Web-Crawler Networks. We apply the technique by combining the domain of load-balancing, with machine learning concepts and Web-Crawler Networks. A prototype algorithm named KALMAN-BLAWLB is designed, and tested in a simulated environment. We measure; how fair the KALMAN-BLAWLB is able to load-balance, system utilization and scalability. KALMAN-BLAWLB outperform all the algorithms that we are able to test in the simulated environment. Finally we conclude that KALMAN-BLAWLB is able to fairly load-balance, achieve a decent system utilization and is scalable, but further tests are needed to confirm large-scale usage.

Preface

This master thesis is submitted in partial fulfillment of the requirements for the degree Master of Science in Information and Communication Technology at the University of Agder, Faculty of Engineering and Science. The project is supported by Integrasco A/S, who has provided data material and equipment for the various simulations performed in this study. This work was carried out under the supervision of Associate Professor Ole-Christoffer Granmo at the University of Agder, Norway.

First I want to thank my supervisor Ole-Christoffer Granmo, for great assistance and inspiration throughout the project period. I also want to thank my co-student Stian Berg for the many fruitful discussions during the project period. Berg have provided me with valuable insight, and without his help I would probably never accomplished all the details described in this thesis. I also owe my boss Jan Hansen and all my colleagues at Integraso A/S, a special thank you for great support and offering me valuable time throughout the project period.

Grimstad, May 2010
Tarjei Romtveit

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

To download a large amount of web-pages, a large scale and distributed web-crawler is typically used. A distributed web-crawler consists of Web-Crawler Agents that can be placed on several locations in a Web-Crawler Network. Load-balancing is conducted by distributing Agent-Tasks, i.e. , representing web-pages, that should be processed by the Web-Crawler Network. Load-balancing is in this setting particularly difficult when introducing factors like varying traffic conditions and changes in the network configuration. This eventually means that the most suitable Web-Crawler Agent for a web-page to get retrieved from is constantly changing i.e non-stationary.

Classical approaches to load-balancing of Web-Crawler Networks have utilized multiple methods, ranging from web-page content analysis, to *consistent hash* functions. The disadvantage with the classical solutions are that they tend to ignore the non-stationary behaviour and utilize complicated location techniques to assign Agent-Tasks in the Web Crawler Network.

In this thesis we present an alternative approach, that uses learning to load-balance a Web-Crawler Network. The novel approach includes to apply a recently published learning technique named BLA-Kalman [23]. We combine the technique of load-balancing and the BLA-Kalman, by relating the load-balancing problem to the non-stationary MANBP. The non-stationary MANBP is the particular problem that BLA-Kalman has been proved to solve. Further, we present a solution in a form of a decentralized prototype algorithm. This algorithm is tested in a simulated web-environment. The parameters measured in the experiments are fair load-balancing and system utilization

1.1 Background and motivation

1.1.1 Web-Crawler Network

Web Crawlers

Large scale retrieval of web data have been utilized by search engines for indexing purposes , i.e. , making the content search-able and let users search in the content downloaded. Examples of search engines are Google, Bing and Yahoo. These are public search engines that try to collect all public information found on the web and expose it to users. [10] [16] In addition to the large search engines there are multiple search engines that index smaller subsets of the web. These subsets may contain a special niche of web pages. Examples are the search engines targeting blogs and discussion forums, Technorati, Omgili and Integrasco Word Of Mouth Search. Most of the specialized and general search engines are backbones for the start-up and success of the companies owning them. In fact the search engine based company Google is ranked as 155 on the Forbes global 2000 ranking in 2009, over the largest companies in the world [21] Centralization of data can therefore be said to be important for the rise of several small and large companies the last decade.

To be able to centralize data, an automatic process that downloads and stores web-pages is necessary. This type of process would have to visit a large amount of web-pages and store content from retrieved data. This niche of processes is called web-crawlers, and several algorithms are suggested the last few decades [8] [10] [32].

Baeza-Yates et al. estimated that it existed above 20 billion indexed web pages in 2007. [4] To retrieve and maintain even a fraction of this amount is highly impractical to do with a single computer. Therefore a distribution of the web-crawler process into multiple Web-Crawler Agents in a Web-Crawler Network is suggested and tested in the literature [15] [8] [38] [10]. The Web-Crawler Network described by Brin et. al was able to visit maximum 100 pages per second, by using four Web-Crawler Agents on various machines [10]. They where able to visit 24 million pages in 10 days, and fetched more than 75 million web-pages during their test-period. One of the first publicly known approaches to distribute a crawler in a Web-Crawler Network, was suggested by Sergey Brin and Lawerence Page in 1998. [10] The distribution was a part of a system called Google, that later became the foundation for the company Google Inc.

As the need have increased the last decade, researches have developed new architectures and methods of distributing Web-Crawlers. One particular field of interest that have gained attention, is that the requests sent and received from the web may travel a long distance. This distance can be caused by the geographical distance between the Web-Crawler Agent and the target host, or other network topological issues. To minimize this distance; Papapetrou et. al suggested a new scheme that distribute the Web-Crawler Agents geographically [40]. The idea is that the Web-Crawler Agent closest to the web-page host should retrieve the web-page. By utilizing this advantage, the Web-Crawler Agent can

retrieve the content faster. However, other and newer approaches want to exploit the available computer resources in a collective effort to gather data to a non-profit search engine [25]. These particular Web-Crawler Networks are called geographical distributed Web-Crawler Networks and would be the particular type of interest in this thesis.

Scaling-out a Web-Crawler

There are discussed two types of general scaling approaches in the literature: scale-out and scale-up [36]. Scaling-out is shortly explained to add processing capacity in a system by adding smaller interconnected computers. Scaling-up is on the other hand deployment of monolithic systems on large Symmetric MultiProcessing computer (SMP)s. Since Web-Crawler Networks consists of several independent processes , i.e. , the Web-Crawler Agents which are able to run in parallel on independent computers, we can classify Web-Crawler Networks in the sense described in [15] and [8] as scale-out systems [10]

Scaling-out architecture have been included as one of the main driving factors behind the cloud-computing trend. Scaling-out in the cloud makes it is possible to dynamically add processing units as the application usage increases. Typical start-up companies like Integrasco can then scale-out its applications only when the customers actually uses it, and save expenses of not buying spare servers. [34] Several companies, like Integrasco, Twitter and LinkedIn, have described their architectures in general. As a common component scaling-out have been utilized in several tiers of their applications to maintain scalability to a moderate cost. Therefore by scaling-out with several cheaper computers have been seen as cost efficient and practical software wise. [52] [53]

The algorithms and architectures utilized in scaled-out systems must be able to handle parallel processing and message handling between the interleaving processes. These requirements have provoked the advent of paradigms (e.g. Map/Reduce, Cloud computing) and scalable systems like Nutch,Hadoop, UbiCrawler, BigTable and UCYMicra. These paradigms and systems utilize clever load-balancing principles, message handling, data consistency and failure detection systems to overcome many of the issues known to cause problems in distributed systems. [36] [17] [34] [20] [8]

Baeza-Yates et al. defines several categories for a Web-Crawler Network where the parallel behaviour influences the algorithm: Partitioning, communication, dependability and external factors. The coarse-grained definitions are further elaborated into several key issues: web-page-assignment, Re-crawling, web-page exchanges and a diversity of external factors. External factors that are mentioned are network topology , i.e. , varying traffic conditions, web growth and Quality Of Service (QOS) of web-servers and Internet connections. This variety of issues leads to several implementations and suggested algorithms in various branches of computer science: consistency, resource allocation, failure detection, scheduling and load-balancing. [4]

1.1.2 Scheduling and load-balancing

Since the need for distributed programming have been increasing the last three decades, efficient resource allocation techniques have been sought to fully utilize the computer hardware [13]. The general scheduling problem have been described a number of times in the literature in the late 70s and early 80s [35], contrary load-balancing have been more extensively studied in the 90s [13]. Further, scheduling and load-balancing have entered niches like Web-Crawler Networks and web-server systems in the 2000s. In addition experiments with cooperative gaming theory to load-balance computer systems, have also been conducted in the 2000s [44]. Despite many years of research, load-balancing is continuously evolving and new niches and prospects are emerging.

Scheduling and allocation

In the article published by Polychronopoulos et. al and the comprehensive taxonomy of scheduling algorithms by Casavant, the main definition of scheduling is to distribute several tasks across processing elements [46] [13]. These elements could in a general sense be both loosely coupled processors in a large network or a single-unit processor capable of process multiple tasks in parallel. Further one or more goals need to be stated by the system creator, i.e. , the objective of the specific method. Examples of goals that are mentioned in the literature are minimizing execution time, minimizing communication delays and maximize resource utilization [6]. These goals could be more problem specific and can include goals that consider optimal routes through a Multiprotocol Label Switching (MPLS) network, geographical dispersion of tasks and fair distribution.[39] [15] [44]

How to define allocation and scheduling have been argued in the literature. Casavant argues that scheduling and allocation is principally two terms describing the same mechanisms [13]. The difference is only the view-point of the method. It is further explained that the allocation view-point is resource centric and the scheduling view-point is consumer centric. This means that allocation is utilization of resources, i.e. . the processors, and scheduling is best utilization for the consumers , i.e. , the tasks.[13] There are several definitions similar to this in the literature. One is the optimization of re-crawling of web-pages described by Granmo et. al , where the resources (i.e the processors) in the web-crawler should only be focused on updated web-pages [24]. This is a typical resource centric scheme where the goal is to avoid resources to be wasted on tasks that not serve the higher goal of the system. Examples of consumer centric schemes are described by Nasri et. al, Penamatsa et. al and Jos'e et. al [38] [44] [18]. These are traditional task assignment schemes and the main question considered is to which processor a specific task should be sent. The progress of processing tasks , i.e. , the consumers, are the main focus in such schemes, therefore seen as consumer centric. We can therefore state that allocation is the decision what tasks that should be assigned for processing and scheduling is to which processor the task should be sent.

Static and dynamic scheduling

In the literature there are mainly two classifications of scheduling algorithms; static and dynamic scheduling. The difference between static and dynamic scheduling is mainly that the tasks are assigned before program execution in static algorithms. Contrary, dynamic algorithms continuously assign tasks to the processors during the system run-time. The consequences of selecting a static scheme, is that it need to predict the future while a dynamic scheme adjusts according to the environment. The dynamic schemes is however known to have some processing overhead, since the calculations need to be performed continuously. Static scheduling is therefore utilized in many early and predictable systems, while dynamic scheduling is utilized in larger and less predictable systems. [44] [6] [13]

Scheduling and load-balancing have, since it is a well studied subject, been a classified and defined in many ways. One of the most comprehensive taxonomy of the scheduling definitions was presented by Casavant [13]. The definitions presented by Casavant viewed scheduling as in a strict hierarchical context and presented the concrete methods by fitting them into this context. Load-balancing is distinguished as a method and defined below the global and dynamic scheduling branch. Later algorithm descriptions, like in [44], classifies their load-balancing algorithm as a static scheduling method. The main argument for this classification is that the algorithm is not reading any run-time state when executed. The scheduling classification of the load-balancing algorithms performed in [13] also implicates that load-balancing is not part of the allocation or resource centric branch of algorithms. It is therefore convenient to state that load-balancing should be seen as a variant of either dynamic and static scheduling.

Load-balancing of Web-Crawler networks

The definition of a task in a Web-Crawler Network, is a web-page that is scheduled to be downloaded by a Web Crawler Agent, and a Web-Crawler Agent is regarded as the processor in the network. The load-balancing mechanism in a Web-Crawler Network is the web-page assignment function. This assignment function is responsible for distributing the tasks of downloading the web-pages, i.e. , the Agent Tasks [38]. The objective is however more problem specific and is not clearly defined in the literature, therefore the load-balancing objective is more specific elaborated by discussing the various Web-Crawler Network objectives mentioned in the literature.

In the load-balancing approaches described by Nasri et. al, Boldi et. al and Hongfei et. al the main objective is to assign the load fairly between the processors in the network [38] [8] [27]. Fairness have also been studied in earlier research within the load-balancing domain, and it is therefore important to distinguish fairness as a common objective to many algorithms. Fairness is usually closely related to system utilization in distributed computing, since the task distribution determines how much work each processor needs to do. Further fairness is relatively trivial to measure by utilization a fairness index as

described by Penmatsa et. al and is therefore regarded as one of the main objectives for load-balancing of Web-Crawler Networks [44].

To minimize geographical dispersion of the Agent Tasks in a wide-spread Web-Crawler Network is described as a web-crawler specific objective [15] [4] [19]. Shortly this minimization is to ensure that the Web-Crawler Agents are downloading web-pages located geographically close to the Web-Crawler Agent. By considering geographical dispersion in the load-balancing algorithm, you gain increasing resistance to network errors and decrease and the global network [15]. An example is if a web-page is hosted on a web-server located in the outskirts of the Internet topology. A typical outskirt can be a web-server connected to a national Internet hub that is poorly interconnected or a faulty Digital Subscriber Line (DSL) connection. When considering such external factors in load-balancing, it is not arbitrary which Web-Crawler Agent is selected to process a specific web-page. It is therefore important that geographical dispersion is considered by the load-balancing process in a Web-Crawler Network.

Re-crawling is in the literature described as the problem of finding the optimal download interval to increase data-consistency between the centralized storage and the web-page. [4] There are suggested several re-crawling policies to find the correct visit interval. These policies utilize various techniques from interpreting Hyper Text Transfer Protocol (HTTP) parameters and learning systems [26] [24]. To find the optimal re-crawling interval could be seen as best use of the resources in a Web-Crawler Network. This resource centric way of thinking could be associated with the definition of allocation argued by Cavasant [13]. It is therefore convenient to see re-crawling as a separate problem branch and not considered within the scope of load-balancing.

In a large Web-Crawler Network failures can and will occur. Examples of relevant failures that can happen are: Web-Crawler Agent crash, loss of Internet connection and overloading hardware resources. Earlier load-balancing algorithms like the one presented by Nasri et. and Hongfei et. al., mention and describe failure tolerant load-balancing schemes, that redistribute the tasks in a Web-Crawler Network when a Web-Crawler Agent crashes [38] [28]. The objective of the redistribution is to fairly distribute based on the current state of the system. Fair redistribution of tasks in a Web-Crawler Network when a failure occurs is therefore seen as an objective for load-balancing algorithms.

1.1.3 Classic solution approaches

There are multiple solution approaches to solve load-balancing in Web-Crawler Network. Each approach aim solve multiple aspects within the load-balancing domain. Examples are approaches that makes load-balancing decisions by taking into account the geographical dispersion of web-pages. Other approaches focuses on the web-page assignment to distribute load fairly in the Web-Crawler Network and other reconfiguration to be resilient to failure. [42] [38] [28]

Geographical distributed scheduling schemes

The load-balancing approaches that mainly focuses on geographical dispersion related issues are utilizing a range of complex location techniques. An example of such approach is the solution suggested in by Papapetrou et. al called IPMicra [41]. Their solution included a a method that tries to find the Web-Crawler Agent most suitable for an Agent-Task by identifying in which Internet Protocol (IP)-subnet the web-page hosts are located. The result of this approach is if a page is located in China, you download it from a Chinese server etc. Another approach that use a much more complex technique was suggested in [19]. The suggested method partitions the Internet topology and applies content analysis to further decide on which partition the web-page is located. Based on this knowledge, it is possible to distribute web-pages to the closest available Web-Crawler Agent.

The suggested methods by by Papapetrou et. al, and Exposto et. al does not consider any load specific parameters in the Web-Crawler network, like CPU and other computer resources. The methods assigns to the best location, topology wise, and therefore ignores to a certain level if the specific Web-Crawler Agent have too many tasks at hand [19] [42]. Another drawback with the method, described Exposto et. al, is the dependency of analysing the downloaded content. By including such a mechanism the method dependent on the feedback-quality and how accurate this technique is. This makes the solutions difficult to measure and adds a new source of error to the algorithm. It is therefore arguably to state that the suggested solutions do not make the necessary trade-offs between geographical dispersion, load etc. and are too dependent on external resources.

Fair assignment schemes

The algorithms suggested in Nasri et. al and Boldi et. al, are utilizing *consistent hashing theory* to assign the web-pages to the Web-Crawler Agents [8] [38] [31]. The consistent hashing theory algorithm, used in the UbiCrawler presented by Boldi et. al, solves the Agent Task assignment without any known global state of the system [8]. The system assigns Agent Tasks fairly across the Web-Crawler Agents, but does not consider the properties of each web-page and what load they eventually creates for each Web-Crawler Agent. The solution presented in Nasri et. al argues that to be able to scale a Web-Crawler Network, you need to add a parameter that identifies the amount of subordinated web-pages, located beneath the assigned web-page [38]. To do this calculation it utilizes the power of law distribution formulae. However both algorithms seem to statically assign web-pages before the Web-Crawler Network starts to operate. This means that the algorithms can not adjust to changes in the Web-Crawler Network, and may not fit into the 'load-balancing' definitions presented by Casavant in [13].

To be able solve all the issues of distributing web-pages in a geographically distributed Web-Crawler Network, this thesis will describe a new and dynamic approach that supports both Internet topology issues and varying capacity issues. The approach utilizes a machine learning technique that is completely novel, and as far as we are aware of no

competing methodology uses such a technique.

1.1.4 Bayesian Learning Automata (BLA) an applicable solution technique?

A machine mechanism designed to follow a predefined pattern of operations, or respond to predefined instructions is called an automation. The concept of LA is defined as a learning automation, where the predefined instructions are referred to as actions, and the LA decides which action to choose based in the response input from the outside environment.[56] LA is a learning technique that is included into the reinforced learning paradigm in machine learning. It is known for its many applications within resource allocation and decision making, and the method have also been utilized to schedule tasks across processors earlier [39].

The k-armed bandit problem have been utilized to model several resource allocation or gaming environments the later years. In the k-armed bandit problem you essentially have a two (or multi-) armed bandit(s) and a player that want to know which arm is the most beneficial. To find the most beneficial arm, the player must balance the exploitation of an arm known to give rewards, versus exploring other arms. If either of the strategies is emphasised too much, a non-optimal or no solution at all may occur. To solve this battle, a Bayesian Learning Automata approach is proposed by Granmo. This approach base itself on prior estimates to select the most beneficial arm and perform exploration. The approach performs better than any previously applied methods to solve the k-armed bandit problem variant Two-Armed Bernoulli Bandit Problem (TABBP). [22]

There a two general variants of the k-armed bandit problem. A stationary variant where the arm feedback probabilities are fixed, and a non-stationary variant where the feedback probabilities change either instant or gradually. In a non-stationary k-armed bandit problem this means that the probability of getting a beneficial pull from a specific arm is not constant. The proposed BLA variant have known issues when interacting with non-stationary k-armed bandit problem. The problem is related to the adaptability of the BLA. Tests shows that the performance of BLA is drastically decreased when applying it to the non-stationary k-armed bandit problem [12]. To solve the performance issues was therefore the subject of Granmo and Berg [23]. They proposed a BLA using Sibling Kalman filters to adapt to non-stationary k-armed bandit problems.They tested the solution by solving a non-stationary version of the k-armed bandit problem called MANBP.

The feedback received from pulling an arm in a MANBP environment, is a normal distributed stochastic value. This means that any problem where you have a normal distributed stochastic value, BLA-Kalman can possibly provide a solution. For instance the process times for each Agent-Tasks in the Web-Crawler Network is a normal distributed stochastic value. The values are stochastic because of the web-environment is stochastic

and it is normal distributed when registered sufficient amount of times. [1]

A Web-Crawler Network domain can therefore be compared with the MANBP domain. This means that with appropriate modifications, it should be possible to load-balance the Web-Crawler Network by applying BLA-Kalman.

We can summarize and state that load-balancing a Web-Crawler Network is analogue to the MANBP presented by Granmo et. al and load-balancing is possible to solve by applying BLA-Kalman [23]. The load-balancing problem is both non-stationary and the feedback values from the Web-Crawler Agents are normal distributed. These similarities are however confirmed on a high-level basis and algorithmic changes would have to be suggested and tested.

1.2 Thesis definition

We formulate the thesis definition in the following fashion:

The purpose of this thesis is to propose how the BLA-Kalman [6] can handle load-balancing in large scale web-crawler networks. To evaluate the performance and scalability empirically, a prototype should be designed and tested in a simulated web-environment. In addition the solution should be compared to exiting methods in the literature that can be tested in a simulated environment. The main effort should be to design and implement a BLA-Kalman solution that is applicable for assigning URLs in a web-crawler network that adapts to non-stationary behaviour.

1.3 Research questions

Is it possible to load-balance a geographical distributed and non-stationary Web-Crawler Network by applying BLA-Kalman ?
The question is an overall problem statement, and covers the central research element, i.e. , to load-balance a Web-Crawler Network. The question also defines a technique that should be applied to the problem and that the scheduling should be performed in a geographically distributed Web-Crawler Network. The technique is the learning algorithm proposed by Granmo and Berg called BLA-Kalman [22]. The BLA-Kalman algorithm is specially designed to filter noise and make accurate decisions in non-stationary environments. The BLA-Kalman algorithm should therefore be modified to load-balance Agent Tasks based on changes in the non-stationary Web-Crawler Network. The proposed algorithm that apply BLA-Kalman to load-balance is called, for the sake of brevity, KALMAN-BLAWLB.

The question is however very general in nature, and does not give an impression of

[1] If we apply the Central Limit Theorem on any large set of retrieved values, it is always normal distributed with a mean and variance [50]

what parameters that need to be measured on the finished KALMAN-BLAWLB solution. We therefore define several subordinated research questions, to give more precise definition of what we want to measure and how we intend to compare the KALMAN-BLAWLB to other existing solutions.

1.3.1 Subordinated research questions

- **In what degree can KALMAN-BLAWLB fairly load-balance Agent-Tasks in a geographically distributed Web-Crawler Network?**
  When load-balancing a Web-Crawler Network it is important that all Web-Crawler Agents receive a limited amount of Agent-Tasks to avoid undesired concentrations of Agent-Tasks. An unwanted concentration can for example occur when Web-Crawler Agent receives a large amount of Agent-Tasks, that take long for it to process. This is even worse in a geographically distributed Web-Crawler Network, when the distance between the host and the Web-Crawler Agent may influence the process time. We therefore have defined a few interesting aspects we intend to observe when measuring fairness:
  - Does the KALMAN-BLAWLB solution keep the Web-Crawler Network fairly load-balanced at the same level over a time-period?
  - How long time do KALMAN-BLAWLB need to learn, before it stabilizes?
  - Do various parameter combinations, passed to the KALMAN-BLAWLB influence the load-balancing?

  In addition the mentioned aspects need to be compared with the same aspects of a few selected competing algorithms.

- **In what degree is the system utilization influenced by using KALMAN-BLAWLB to load-balance a geographically distributed Web-Crawler Network?**
  Even if the KALMAN-BLAWLB is avoiding unwanted concentrations of Agent-Tasks, the overall system utilization may not be better compared with other solution approaches. System utilization is typically mentioned in the literature as Agent-Tasks processed per second [44]. A greedy solution could easily assign a lot of Agent-Tasks to a Web-Crawler Agent that is really fast. This manages to keep the overall system utilization up, but Agent-Tasks may be processed in an non-beneficial manner that is not utilizing all the capacity. It is therefore interesting to see how the system utilization is influenced by the KALMAN-BLAWLB compared with a few competing algorithms.
  - Does the KALMAN-BLAWLB succeed to stabilize the system-utilization?
  - Does the KALMAN-BLAWLB outperform the the competing algorithms?
  - How is the system utilization compared with the fairness?
  - Do various parameter combinations, passed to the KALMAN-BLAWLB influence the system-utilization?

- **In what degree is the load-balancing and system utilization influenced when scaling up the amount of Agent-Tasks in the Web-Crawler Network?**
  When considering that the solution should be possible to deploy in a realistic environment at a later occasion, it is interesting to observe how the solution behave when adding more Agent-Tasks. This is seen as the first step towards scaling the solution to a realistic environment. To therefore identify the scalability, we observe both fair load-balancing and system utilization and answer the following:
  - How is KALMAN-BLAWLB able to stabilize the load-balancing and system utilization when increasing the amount of Agent-Tasks?
  - How is the learning influenced when KALMAN-BLAWLB need handle more Agent-Task?
  - Do various parameter combinations, passed to the KALMAN-BLAWLB perform differently than before scaling up the amount of Agent-Tasks?

1.4 Claim

In this thesis we claim that load-balancing of Web-Crawler Networks and the non-stationary MANBP clearly have several similarities. These similarities is claimed to be that the Web-Crawler Agents are perceived as arms on a k-armed bandit and each Agent-Task perceived as a bandit player. Further, we claim that load can be measured as a normal distributed value, which includes the time it need to wait in the process queue and the RTT between the Web-Crawler Agent and the web-host. This normal distributed value we then claim to be a suitable feedback value associated with the selection of a Web-Crawler Agent. The mean and variance of this value varies in addition by time, and we claim that the Web-Crawler Network is non-stationary. The main claim is therefore that the BLA-Kalman can be utilized to load-balance a non-stationary geographically distributed Web-Crawler Network.

When considering empirically measured results, we claimed that fair load-balancing is the most important goal to achieve for the algorithm. In addition we claim that the solution utilizing BLA-Kalman achieve a greater degree of fair load-balancing than consistent hashing techniques and IPMicra. We further claim that the system utilization is considerably increased by using the BLA-Kalman solution, compared with consistent hashing and IPMicra. Lastly, we claim that the solution is scalable, but the complexity of the load-balancing is increasing when adding more Agent-Tasks to the Web-Crawler Network.

1.5 Contributions

The main contribution of research in this thesis, is a prototype algorithm called KALMAN-BLAWLB, that is tested in a simulated web-environment. This algorithm is a decentralized load-balancing algorithm, that utilize the BLA-Kalman to distribute Agent-Tasks

to several Web-Crawler Agents located in a Web-Crawler Network. To our knowledge, KALMAN-BLAWLB is the first known application of the BLA-Kalman algorithm for a purpose like load-balancing. It is also one of few learning based approaches within the whole load-balancing and scheduling domain.

To be able to create KALMAN-BLAWLB we contribute a scheme that merges the domain of load-balancing Web-Crawler Networks and the MANBP. This is necessary to be able to find the role of the BLA-Kalman and identify what feedback it should receive. To be able to create this scheme the basis problem of scheduling and load-balancing is elaborated. Next, the basis problem and the load-balancing problem is properly defined, before drawing the scheme as a result of the definitions. Lastly, the scheme contributed is classified as a multi-agent environment, in addition to be non-stationary and stochastic.

One of the contributions, which appeared from the problem solution, is the load measurement: Process time . This process time is used as feedback to the BLA-Kalman algorithm. This process time includes the amount of time each Agent-Tasks need to wait and the RTT between the Web-Crawler Agent and the web-page host. This process time is therefore a simplistic load measurement, that extends further use of the KALMAN-BLAWLB. For example if deciding to use another allocation scheme than tested in the thesis, the KALMAN-BLAWLB can be extended without modifying the load-balancing characteristic.

The results contributed in this thesis show that KALMAN-BLAWLB is able to fairly load-balance a geographically distributed Web-Crawler Networks and obtain a reasonably good system utilization. We believe that the results we provide give an good insight in usage of BLA-Kalman in load-balancing applications and inspire further research in the domain of load-balancing Web-Crawler Networks.

1.6 Target Audience

The target audience of this thesis, is anyone that have interests within the scheduling and load-balancing domains. Particularly, people that is interested in load-balancing Web-Crawler Networks. However, since the main contribution of the thesis is to apply a machine-learning technique called BLA-Kalman on the load-balancing problem, AI researches and other people interesting in this field may find the thesis interesting.

For the reader to fully understand the concepts and reasoning behind the solution, some knowledge of machine learning and a good understanding of common basic elements within the field of computer science is recommended. In addition there are several elements of probability theory mentioned in the sections explaining BLA-Kalman. This means the reader should have some fundamental knowledge of probability theory, since we not intend to give a comprehensive introduction to probability theory in the thesis.

1.7 Report Outline

The rest of this thesis is organized as follows: Chapter 2 introduces the general concepts of scheduling and look at several classifications, including load-balancing. Chapter 2 also tries to clear some term confusions within the scheduling domain. Chapter 3 introduces the Web-Crawler domain in a historical perspective, the general behaviour, architecture etc., in addition to a few allocation and load-balancing algorithms. Chapter 4 introduces briefly concepts behind machine learning, in addition to the BLA and BLA-Kalman algorithms . Chapter 5 is intended to present problem definitions formally. It also explains how we want combine the domain of load-balancing Web-Crawler Networks with the BLA-Kalman algorithm. In Chapter 6 the KALMAN-BLAWLB is thoroughly explained by the main architectural concepts and a detailed algorithm design. In Chapter 7 sufficient background and details for the test-setup are provided, in addition to how we designed the simulation environment. Further, the results obtained from the experiments are presented as commented plots. In chapter 8 we discuss our main findings and some additional aspects of the KALMAN-BLAWLB. Chapter 9 is intended to wrap up, provide a conclusion and suggest interesting aspects that may be pursued in further research.

Chapter 2 ,3 and 4 are intended to supply background information and also to be independent from each other. However some concepts, like scheduling and load-balancing, is found in several of the chapters. Therefore it is recommended that these chapters is read chronologically order. Chapter 5 to 8 should however be read in chronologically order to get the whole picture.

Chapter 2

Scheduling and load-balancing

Chapter 2 give a brief overview of the scheduling domain. It describes several categories of scheduling algorithms and try to clear some term confusions that often occur in the literature. Further the chapter details the formal problem related to scheduling and load-balancing problem, and describe the computational complexity associated with it.

2.1 General scheduling definitions

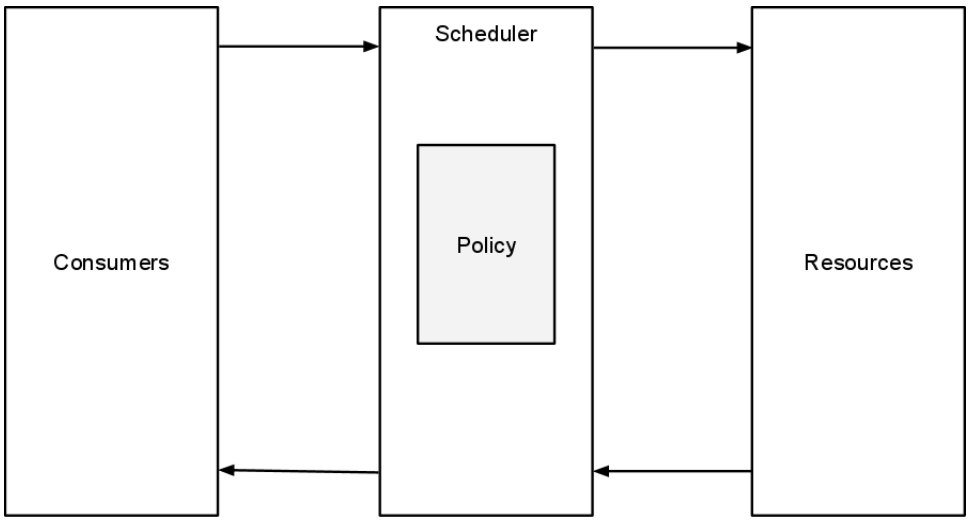


Figure 2.1: The figure shows the four essential components in a resource allocation/scheduling problem: Consumer(s), Resource(s), Scheduler and Policy. The Policy describes how the Scheduler is interacting with the Resource(s) and Consumer(s). The Consumer(s) can be defined as the tasks, web-pages, calculation etc. that needs to be processed. The Resource(s) is then seen as the processing elements (e.g. CPU, computers etc.)

The taxonomy performed by Casavant, defines scheduling as the distribution of tasks between several processors [13]. This definition is further described by identifying several core components in any scheduling problem. The three main components Consumer(s), Resource(s) and Policy are shown in the scheduler context in figure 2.1. Consumers are seen as the tasks (e.g web-pages, operating system processes etc.) and the Resources are seen as the processing elements (e.g CPU, network-nodes etc) in the scheduling system. The definition of Policy is further explained as how the Resources or/and the Consumers are handled in the scheduling process. It is further argued that the Policy must have certain goals to fulfil. These goals can be minimizing execution time, minimizing communication delays and/or maximizing resource utilization. It is therefore convenient to assume that scheduling in general can be defined as follows: Distribute tasks among processors to achieve some performance goal(s). [13] [6]

Scheduling have been defined on several levels the last three decades. An thorough hierarchical classification is performed by Casavant in [13] and the hierarchy tree is shown in figure 2.2. The two root branches in the tree are local and global scheduling. These branches describe the essential two types of scheduling algorithms: Local operative-system scheduling and distributed scheduling in a global network of processors. Local scheduling is more precisely defined to give time-slices to to a single processor by the operating system. This branch seen as the lowest level of scheduling algorithms and not further considered when classifying algorithms. This because such algorithms are considered to be contained in standalone computer system and not in distributed systems. Classification of scheduling in distributed systems was the main effort of Casavant and is also considered be the most relevant domain for this thesis [13].

2.1.1 Allocation vs. scheduling

Since it is two roles in the system, i.e. , the Consumer(s) and the Resource(s), the Policy can have two view-points of its operations: Resource centric and Consumer centric. This is argued by Cavasant in [13] as the difference between allocation and scheduling. Allocation and scheduling is by Casavant considered as two terms commonly found in the literature, with ambiguous definitions. Resource centric view is by Casavant seen as best utilization of the CPUs, processing nodes etc. and the main view of allocation schemes. The Consumer centric view is however seen as how to assign resources to the Consumers most efficiently and the main view of scheduling algorithms. Allocation schemes may focus on the specific order of the tasks, instead of the distributive mechanisms. An allocation scheme may organize a set of tasks efficiently in a Directed Acyclic Graph (DAG) or use a learning algorithm to assign resources dynamically [47] [24]. It is therefore to convenient to state that scheduling and allocation are within the same domain but with separate operation view-points. Scheduling and allocation is in this thesis treated as two algorithms and the main focus is on scheduling. Intersections between allocation and scheduling algorithms are however acknowledged and explained.

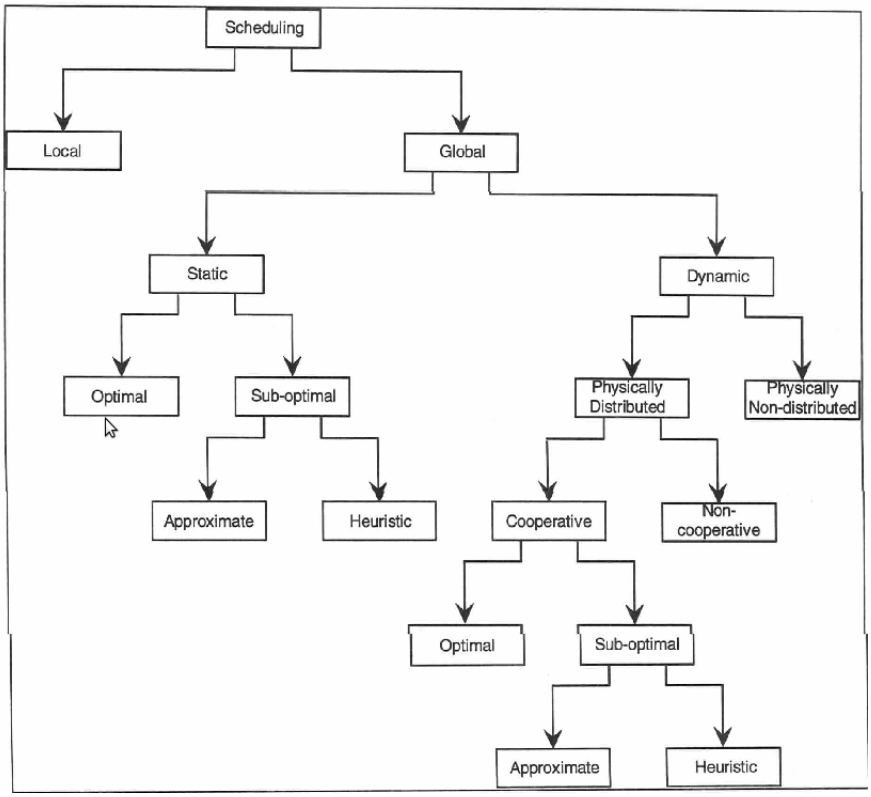


Figure 2.2: The figure shows the hierarchy of scheduling algorithms. The two main branches of algorithms are local and global scheduling of processes. Local scheduling is low-level scheduling on single processor and global scheduling is the overall scheduling of tasks between several processor in a larger system. The figure shows further taxonomy of the various types of algorithms and describes the modes of operation. [13]

2.1.2 Static scheduling

In figure 2.2 the static scheduling algorithm is placed beneath global scheduling. Static scheduling is shortly explained to pre-distribute the tasks before the tasks are executed on the processors. This is often referred to as to distribute tasks compile-time. Static scheduling is seen as non-preemptive, since the processes are executed in the specific order and location assigned by the scheduling algorithm [6]. There are several examples of static scheduling algorithms that pre-distribute tasks in a network of processors. Some of these are developed in the early years of distributed systems, like the algorithms analysed and described by Virginia Mary Lo and Shirazi et. al [33] [54]. In addition there are new and interesting static scheduling algorithms described by Panmatsa et. al and Nasri et. al [44] [38]. The scheduling algorithm described by Penmatsa et. al is even classified as a load-balancing algorithm. In earlier classifications by Casavant, load-balancing algorithms are considered to be below the dynamic scheduling branch. This is intuitive, since a load-balancing algorithm should continuously adjust load in a system of multiple

processors. Static scheduling algorithms are therefore pre-distributed schemes that is by Casavant not seen as load-balancing schemes. However the term static load-balancing algorithm are mentioned by many, but seen as a bit counter intuitive and not utilized in further definitions. [13]

The large advantage of static scheduling methods is that there are little or no overhead during the run-time of the algorithm. This is caused by pre-calculations performed in advance. The disadvantage compared with the dynamic algorithms, is that the static algorithms need to predict the future. Dynamic algorithms may adjust the scheduling according to changing parameters, caused by factors that is impossible to determine in advance. Examples of factors are varying processing speeds on separate tasks and hardware failure. Therefore are static algorithms most suitable for systems that contain predicable tasks and have few failures. [13] [6]

2.1.3 Dynamic scheduling

The dynamic branch of algorithms distinguish itself from the static, by performing the task assignment during the task execution. This is often referred to as performing the assignments run-time [13] [3]. The main advantage of using a dynamic scheduling algorithm contra a static, is the adaptability you get if you are trying to schedule in a distributed system. These systems consists of many various types of processors (e.g. computers,CPU) and therefore have varying processing speeds and operative environment. A dynamic scheduling algorithm can be utilized to sense the differences in the operative environment and change the scheduling policy in the system. Multiple dynamic algorithms are described by Hamidzadeh et. al in [3]. However the disadvantage of these algorithms are that they continuously need to operate and consume resources from the processors. Dynamic scheduling algorithms, may therefore not be suitable for systems that require a high degree of processor utilization.

A dynamic scheduling algorithm can be decentralized [1] or centralized. This is referred in figure 2.2 as physically distributed and non-physically distributed. The difference is described as where the responsibility of the scheduling is taking place. In decentralized algorithms, the decision of where to process the tasks is calculated by the processors. Contrary; in centralized algorithms the decisions are performed on a central processor. A centralized algorithm may need to decide based on some shared global state. However many such algorithms have a single point of failure, since the decisions need to be conducted by one unit. The decentralized algorithms do not have a single point of failure, but the performance of the algorithm may be degraded because there is lack of global knowledge. Global knowledge that could be missing is a comprehensive register of the total system load or variables related to a special group of tasks. When such global knowledge is missing, the decisions in the distributed algorithm may be less accurate than in a centralized algorithm. [13]

[1] The term decentralized is referred by Casavant as distributed.

In the category of distributed scheduling algorithms two subordinated categories are defined: cooperative and non-cooperative. Cooperative algorithms are defined by Casavant as algorithms that react based on decisions performed in other parts of the system. A such cooperative system may decide the tasks assignment based on some information modified by the algorithm on another host. This information may be globally shared information or job specific information. Typical job specific information is the last known processing time or amount of Input and Output (I/O) calls during execution. The selected information is however algorithm specific and should include information that serves the goal of the algorithm in the system , e.g. , minimizing execution time, minimizing communication delays and/or maximizing resource utilization. Non-cooperative algorithms do not consider the shared information when making decisions, and therefore may do decisions that influence the other processors in the system in an unpredictable manner. A non-cooperative algorithm would therefore in many cases have low-level goals and not be accustomed to optimize or sub-optimize the scheduling.

2.1.4 Optimal vs sub-optimal scheduling

An optimal scheduling performed by a static or a dynamic approach, is optimal in the sense of finding the best order of tasks and fulfill the goals or a criterion function of the specific method [13]. As an example we assume that you have n tasks with varying execution times. These are to be scheduled on two processors, P_1 and P_2. The goal of the method is to make sure that the overall completion time is minimized. If an optimal schedule is found, the n tasks are processed by fully utilizing both P_1 and P_2 and completed on minimal amount of time. Contrary a non-optimal method with the same goals, allow a certain constraint slack. In the example with n tasks, such constraints may be below maximum utilization and above minimum completion time. Both suboptimal and optimal types of algorithms are classified by Casavant , but it is argued that optimal solutions may be computational infeasible because of the complexity of the problem [13] [6]. This also complies with the general complexity of scheduling described by Bauke et. al where the optimal search in a large solution space is considered almost impossible [5].

Casavant defines, as shown in the hierarchy tree in figure 2.2, two types of solution approaches to sub-optimal dynamic and static scheduling. These are either approximate or heuristic. The difference is shortly that approximate have a measurable objective function and heuristics have not. Further are two factors described that must be considered when identifying if it is an approximate approach: An available metric to evaluate the solution and the time required to evaluate a solution. Heuristics are however solution approaches that have no built in quantitative measurable functions. A heuristic solution is argued by Casavant to be built up of several logical steps that seem intuitively correct and give a 'good' solution to the problem. Pearl and Judea are defining more precisely that heuristics are strategies using accessible information in problem solving in both human beings and machines. This information may be loosely applicable, but can provide some knowledge for further decisions that the system needs to do [43]. An example of a heuristic employed within the static scheduling domain is the Genetic Algorithm (GA). A

GA is applied to schedule tasks in computational grids [1]. This algorithm apply various genetic operators to gradually evolve a sub-optimal schedule. The genetic operators are modified to approximate a continuously better solution through iterations.

2.1.5 Load-balancing

The essentials of load-balancing is redistribution of tasks between processors to fairly load-balance the system. Fair load-balancing of distributed systems have been the focus of many researches the last thirty years, and the main focus are to avoid too much restrain and idle time on the computer-hardware [13]. An example may be that a single processor is heavy loaded by many large scheduled tasks. Another processor in the same system, may have much idle capacity and therefore should relieve the heavy loaded processor. A load-balancing algorithm should therefore recognize such a situation and make an effort to load-balance the system more fairly.

Classification

Load-balancing is an algorithm class, associated with both dynamic and static scheduling. Load-balancing is by Casavant , classified to be between the optimal and suboptimal branches of dynamic cooperative scheduling [13]. This essentially means that load-balancing algorithms must be able to change the scheduling run-time and cooperate its decisions. Further Casavant argues that load-balancing have the same basis problem as the general scheduling problem described in section 2.2. This means that both suboptimal and optimal load-balancing algorithms can exist dependent on a valid method to measure the solution. Other and newer approaches classifies their load-balancing algorithms as static, because the scheduling decision is performed compile-time [44]. This may however be a bit counter intuitive when you need to balance a system by considering the state of the system. The state would in all realistic system change during the run-time and therefore be important to consider when changing the task assignments. All the various meanings and definitions makes the load-balancing term ambiguous and confusing, but the dynamic cooperative definition seem more intuitively correct and is the preferred defenition of load-balancing used in this thesis.

2.2 Computational complexity

To define the scheduling problem more precisely and derive complexity assumptions, it is convenient to map it to a well known problem in computer science. The similarities between Number Partitioning Problem (NPP) is and scheduling is mentioned several places in the literature as the theoretical definition of the scheduling problem [35] [6] [5]. The NPP is a combinatorial optimization problem where the solver wants to partition a set of items, where each item representing an integer or any positive real number. There can be

multiple partitions and the goal of the solution is that each partition is as perfect as possible. Perfect is defined to be when each partition sum is equal to the other partition sums. [9] [35]. The remaining difference between the partitions is in the literature somtimes referred to as the $discrepancy$. The NPP definition may be mapped to the scheduling problem by defining a set of n tasks T in $T = \{T_1, T_2 \dots T_n\}$. Each task T_n is not an integer or a real number, so a β function is defined. This function can for instance return the processing time or amount of operations per T. If we have two processes P_1 and P_2, and we want to distribute the tasks T_n. The set T must be partitioned into the sets T_{P_1} and T_{P_2}. The partitions T_{P_1} and T_{P_2} represents the scheduled tasks on the particular processor P_m. If we assume that the processors have the identical specifications, the content of each partition must be as equal as possible to obtain the example goal of minimizing execution time. This is shown generally in equation 2.1. The NPP problem can therefore be utilized as a reference problem for scheduling and defining computational complexity.

$$
\begin{array}{lcl}
m & = & \text{The amount of partitions} \\
T_i & = & \text{A task} \\
T_{P_i} & = & \text{A partition of tasks} \\
\beta(T) & = & \text{The processing time or operations per second}
\end{array}
$$

$$
\sum_{T \in T_{P_1}} \beta(T) \;=\; \sum_{T \in T_{P_2}} \beta(T) \;=\; \dots \;=\; \sum_{T \in T_{P_m}} \beta(T) \tag{2.1}
$$

The complexity of scheduling may vary with the goal of the problem that is needed to be solved. However a general complexity may be determined if we map the problem to the general NPP. The NPP is one of Garey and Johnsons six basic NP-hard problems, and is the only problem involving numbers [35]. The computational complexity of the NPP is however varying with the characteristics of the numbers found in the set that needs to be partitioned. The NP-hardness of the NPP is established first when the amount of numbers that needs to be partitioned are exponentially large [35]. This essentially means that there exists no algorithms that are significantly faster than full search through the solution space , i.e. , exponential search. The 'hardness' is however dependent on the size of numbers in the problem. Experiments shows that random generated instances of NP-hard problems are relatively easy to solve without exponential search [5].

Chapter 3

Web-Crawlers and Web-Crawler Networks

Chapter 3 give a brief overview over Web-Crawler and the Web-Crawler Networks. It starts to give a small historical summary, before the a typical architecture of a general purpose and distributed Web-Crawler Network is explained. Further a couple types of geographically distributed Web-Crawler Networks is explained. To keep a relation to chapter 2, a brief overview is presented of a few allocation schemes and the relation between allocation and the load-balancing algorithms. The most important section is placed last in the chapter. It describes and classifies the algorithms we intend to implement and compare with the KALMAN-BLAWLB.

3.1 General behaviour and architecture

3.1.1 Historical perspective

Internet, and especially the World Wide Web (WWW), consists mainly of information scattered on many web-servers all over the world. This de-centralization causes the usability to be rather low, since it is difficult to find the exact desired information when no central yellow pages registry, i.e., an index, is found. A such yellow pages registry could be open for the public to search in, or used to provide business intelligence. To be able to create an yellow pages registry of the WWW, a registration process need to be conducted. All the web-pages that is reachable in the public WWW is a potential target for a registration process. This type of large scale registration process may be too difficult to conduct by humans so therefore a process called Web-Crawler , i.e. , sometimes referred to as a web-spider etc, is suggested in the literature. [49] [14] [15][10].

There have been conducted research on Web-Crawlers since the early beginnings of the Internet and World Wide Web. One of the first Web-Crawler was published by Eichmann in 94 [49]. This simple Web-Crawler consisted of a program that allocated and

scheduled which web-pages that needed to be downloaded. In addition a modified American Standard Code for Information Interchange (ASCII) WWW browser was responsible for downloading and storing web-pages. Along with a couple of later approaches, the Web-Crawler process was a relatively simple process, capable of downloading pages from a single computer in a non-distributed fashion. Among the first that acknowledged that the WWW was growing too large for a single machine crawler, was Brin and Sergey. They therefore published a Web-Crawler that was distributed on several machines in a Web-Crawler Network as a part of their Google system

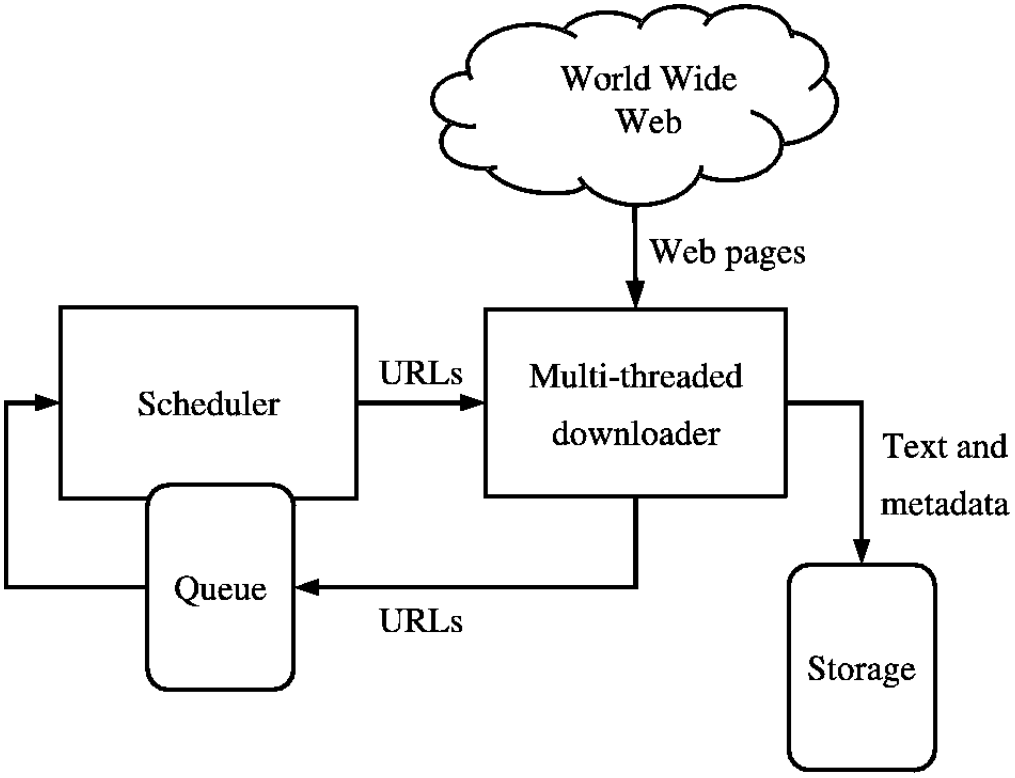


Figure 3.1: A general purpose Web-Crawler architecture showing the core components: Scheduler, Queue, Storage, and the multithreaded downloader. The scheduler is mainly responsible for distributing URLs to the multi-threaded downloaders. The scheduler is together with the queue also responsible for how resources should be allocated to the URLs. The storage component is containing the downloaded documents. The format on the storage, may be a relational database or an index. [14] (Figure 2.10, page 35)

3.1.2 Distributed Web-Crawler Networks

The Web-Crawler components described in the early research by Eichmann did not have any well defined architecture [49]. The Web-Crawler by Pinkerton released in 94, had a more structural approach using a search engine to feed URLs to the various crawler

processes. [45]. The approach did however not generalize a structure, that separated allocation, scheduling and the storage process . This separation is seen in the distributed approach by Brin and Sergey. They utilized an URL-Server to handle the allocation and scheduling of Agent-Tasks to the Web-Crawler Agents. In addition an advanced storage component that index and order pages by utilizing clever page ranking algorithms etc is defined. A further abstraction of the the previous approaches is explained by Castillo in [14]. This abstraction contains the a combined Scheduler and Queue to distribute and allocate resources to the Agent Tasks in the systems. These resources are either non-distributed or distributed multi-treaded Web-Crawler Agents, that retrieves and stores the web-pages in a storage component. A such high-level architecture is shown in figure 3.1.

3.1.3 Geographically distributed Web-Crawler Networks

When you have a distributed Web-Crawler Network, as the one suggested in [40], it is possible to place the Web-Crawler Agents on several locations in the web. An example could be to have Web-Crawler Agents placed in each of the European or American countries. The network latency would in many cases be lower between the web-server and a Web-Crawler Agent placed in for instance the same country. This nearness is however mentioned to be more complex. Nearness is in many geographical aware schemes in terms of network distance (latency) and not in terms of physical (geographical) distance. This means that a web-page may be located in the country Sweden, but be closer in network distance to a Web-Crawler Agent located in Norway. This network distance is therefore found interesting by researches to exploit and therefore the newest addition to the Web-Crawler Network architectures, is the geographical distributed. [40]

Another type of distributed system have gained attention with the advent of non-dial-up internet connections. These systems are made to exploit the fact that there are a large amount of computers present in the world that are connected to the Internet all the time and not fully utilize the bandwidth or CPU cycles. These systems are mainly based on that the machine owners voluntarily installs a client on the computer that give access to some kind of central control. Then for instance weather data calculation tasks etc. are sent to utilize CPU cycles on the computer. A concrete such project, is the Grub project. This geographical distributed Web-Crawler Network, intend to manage a large amount of mobile Web-Crawler Agents that is distributed on voluntarily installed client machines. [7] [25]

3.1.4 Scheduling and Allocation

As mentioned in the introduction section 1.1.2, there are two high-level scheme classifications within the domain of scheduling. These are called Resource(s) centric and Consumer(s) centric scheduling schemes. Allocation , i.e. , Resource(s) centric , schemes focus on most effective usage of the processors. An example of ineffective usage of processors, is that a Web-Crawler download a web-page that is already retrieved earlier.

Scheduling ,i.e., Consumer(s) centric, schemes however focus on processing tasks effectively [13]. It is convenient to similar distinguish the schemes that is published in the Web-Crawler literature. However, most scheduling algorithms related to Web-Crawler Networks are classified as load-balancing schemes. Therefore a few examples of allocation and load-balancing schemes are further explained below.

3.2 Allocation schemes

In many allocation problems mentioned in the literature, the processing sequence is important [29] [47]. When processing sequence is considered, it is important that a task is processed in a specific order. This order is often important when distributing tasks that are parts of a transaction. The precedence-relations between the tasks are often modeled as a DAG, that explicitly defines a sequence of the tasks. In an allocation algorithm in a Web-Crawler Network the precedence-relations may also be taken into account. A structure of a web-page can also be modeled as a DAG, since each page is linked together [57]. Each link could be perceived as an arc and each static web-page can be seen as a node in a DAG [11]. This DAG structure, can therefore be utilized to allocate in which sequence the web-pages should be downloaded.

The allocation algorithms applied to query a typical DAG structured web-page, is for instance the breadth-first and depth-first search algorithms. In the first Web-Crawler described by Eichmann , these two algorithms was his basic approaches to traverse web-pages [49]. They ensured that all web-pages was visited and together with an effective data structure, ensured that no web-pages was downloaded twice in the same crawler operation. However as mentioned in [11], there is a continuously larger portion of the web-pages are not strongly connected in a DAG fashion. This is caused by an increasing amount of hidden web-pages, that needs to be visited directly or are protected from being viewed by some authentication or input from the user. These hidden pages may not have a root node, i.e. , a web-page linking to it, and may only link to itself. Several classifications of the problems involved in how to allocate resources to hidden web-pages are suggest in the paper Crawling The Hidden Web by Raghavan et. al [48].

Several classifications are made by Raghavan et. al of various types of hidden content [48]. The three classifications was client-based , input generated and temporal content. Client-based content are web-pages that are generated for a specific client. This can be a personalized page, the users have customized and need to be logged in to view. Input generated content, is seen as web-page generated by some kind of input from the user. Example of a such web-page, is a site generated when a user inputs a search query, and the results are displayed in the returned web-page. Temporal content, is continuously updated content, that varies with time. Examples may be a web-page following the stock-market or a page displaying news headlines. Since each of the classified content types have independent properties and request models, they need their own allocation schemes that generate requests. [48]

To solve temporal content challenges, a Web-Crawler or a Web-Crawler Network need to have a strategy to find the most ideal revisit interval. These schemes are often referred to as re-crawling schemes. A re-crawling scheme is an allocation scheme that specifically want to assign more or less resources to a specific web-page and prioritize the retrieval rate according to the allocated resources. The main objective is to be able to increase the level of consistency, i.e. , the freshness, between the temporal data found on the web-servers and the storage component. [48] [4]

There are several re-crawling schemes suggested in the literature. One is the adaptive solution suggested by Granmo et. al [24]. This is an solution that apply an LA to allocate resources to each web-page dynamically. The problem is modeled as knapsack problem, were the knapsack contain the resources available in the Web-Crawler. The LA receives a feedback from the Web-Crawler, if the page is updated or not. Based on this feedback it assigns more or less resources, within the constraints of the knapsack. This solution is a very novel approach, compared with earlier solutions that use mechanism present on web-servers. These algorithms depends on that both the web-server and Web-Crawler cooperate [26]. A cooperation scheme needs to be agreed on a world-wide basis, and a such agreement is in many cases difficult to create .

3.3 Load-balancing algorithms

The problem of distributing Agent Tasks on multiple Web-Crawler Agents is inherited from the multiprocessor scheduling problem explained in section 2.1. The problem , described as URL-assignment, is also described by Baeza et. al in [4] as one of the problems that need to be solved in a distributed Web-Crawler Network. The problem is however little discussed in the literature related to Web-Crawlers until the release of the UbiCrawler by Boldi et. al [8] [14]. [1] A couple of relevant scheduling schemes are therefore discussed and classified in section below.

3.3.1 Consistent hashing

Consistent hashing is a technique that was first suggested by Karger et. al. to decrease the amounts of so called 'hot spots' in a large distributed web-server networks [31]. A 'hot spot' is defined by Karger et. al as a node that is suddenly overloaded with requests. A large amount of requests could for instance be targeted at the same web-page and then overload the web-server. To avoid this traffic congestion, a cache protocol is suggested to guide requests fairly across several nodes. This cache protocol is using the consistent hashing mechanisms to decide where to route the request and load-balance the network. [31]

[1] The paper Internet Archive Crawler by Burner is not publicly available at the time this thesis, and the potential relevant scheduling scheme(s) described in this paper is therefore not mentioned

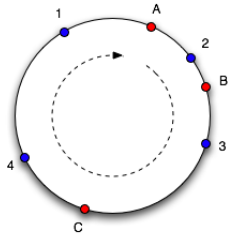


Figure 3.2: The figure shows a consistent hashing circle. The placement of each point is calculated with a hash function returning a number within value-scale of the circle. Typically an integer. The points A, B, C symbolize nodes, i.e. , the Web-Crawler Agents, and the points 1,2,3,4 symbolize objects, i.e. , the Agent Tasks. The Agent Tasks are assigned to first placed Web-Crawler Agent if you follow the circle clockwise from the Agent Task. This means that 1 and 4 are assigned to A, 2 assigned to B etc. The spaces between each Web-Crawler Agent in the circle is sometimes referred to as buckets.

How it works

Consistent hashing is a simple technique that uses a hash function to map a node[1] to a URL, or any object supported by a hash function. The consistent hash function is seen as a successor of the static hash function. A static hash function would hash any object directly to a node without consider a node may fail or maybe removed. If for example the simple hash function $u \bmod 5$ is applied and the object input is $u = 1$. The result would of this calculation be node 1. The problem is that node 1 has earlier been removed from the system, and therefore unreachable. The consistent hash function solves this problem by mapping the objects and nodes in a circle of buckets as displayed in figure 3.2. The circle is read clockwise, and the node locations, i.e. , which bucket to be placed in, are previous calculated by a hash function. To identify which node the object belongs to, the object is hashed and the result is compared with the closest available node in the circle (read clockwise). An example is shown in figure 3.2, where the nodes $\{A, B, C\}$ is mapped to the circle. The objects $\{1, 2, 3, 4\}$ is present and objects 1 and 4 belongs to node A, 2 to B etc. If A is removed , the circle is read clockwise and the objects, i.e. , 4 and 1, now belongs to node B. Therefore consistent hashing can be said to be a failure resistant hashing mechanism, and can be utilized to calculate distribution of multiple objects between nodes. [31] [8] [38]

A problem with the consistent hashing function is that a node may get assigned unfairly many objects. This occurs when the bucket between the nodes in the circle is too large. An example of a large bucket can be seen in figure 3.2, where the bucket between A and C are relatively large. This bucket may contain many objects compared with the bucket between A and B, thus give an unfair balance of objects. To solve this issue it is suggested by Karger et. al , to add replica nodes, of the already existing nodes [31].

[1] A node is sometimes referred to as an agent in the literature.

If we consider the example displayed in figure 3.2, 10 new replicas of each node could be placed in the circle by hashing variants of the node identifications. A identification variant could be $A1 \ldots A10$ etc. By performing this replication it reduces the bucket size and then assigns more fairly objects to each of the nodes. It is measured by Boldi et. al , that the perfect load-balancing is deviated by less than 6% by replicating each node 100 times [8].

Load-balancing Web-Crawler Networks with consistent hashing

In early Web-Crawler Network designs, as in the one published by Cho et. al, a static hash function was applied to assign Agent-Tasks , i.e. , load-balancing, to Web-Crawler Agents [15]. This static hash function performed reasonably well, but had some weaknesses when the global state of the system changed. Most notable weakness is if one of the Web-Crawler Agents in the system failed. To solve these issues it was therefore suggested by Boldi et. al, to use consistent hashing instead of static hashing to assign Agent Tasks in Web-Crawler Networks [38] [8].

To be able to use consistent hashing in a Web-Crawler Network a definition of the node component need to be present. In both the UbiCrawler and the consistent hash assignment function defined by Nasri et. al, a Web-Crawler Agent is treated as a node. This node could be defined by a unique single string identifier or an integer. The Web-Crawler Agents in the consistent hashing circle is however considered as alive agents and part of a larger set of potentially active Web-Crawler Agents. These definitions are intuitively easy to grasp, since Web-Crawler Agents may fail, but be present at later occasion.

Agent Task representation

In a consistent hash circle, an object is what you will attach to each node by hashing its unique identifier. Since it the main objective in load-balancing algorithms is to distribute Agent-Tasks between several Web-Crawler Agents , it is convenient to interpret each object as an Agent-Task. The definitions of what each of these Agent Tasks really represent, is in the solutions by Nasri et. al and Boldi et. al both seen as host names, e.g. , google.com, uia.no, integrasco.no etc. When each Web-Crawler Agent then receives an Agent Task, it is visiting for instance a root page to the specified host. The allocation function residing on the Web-Crawler Agent decides what it further wants to do with the host. This could for instance be to visit the subordinated web-page structure in a particular order or just process the first page. The alternative to a host based representation, is to let each Agent Task represent single web-page. This type of representation is not discussed by either Nasri et. al or Boldi et. al. It is however considered as a good practise to not overload a singe host by executing many requests from the Web-Crawler Agents. It could be possible to execute requests towards the same host, at the same time, on various Web-Crawler Agents, if each Agent-Task represent an unique web-page, i.e. , an URL. It is also stated by Cho et. al that a web-page representative scheme, would generate many

Agent Task exchanges between the Web-Crawler Agents [15]. It is therefore convenient to acknowledge that each Agent Task represent a host identifier in a consistent hashing scheme. [38] [8]

Classifications

The approaches to schedule Web-Crawler Networks that utilize consisting hashing techniques are referred to as load-balancing algorithms by their authors [38] [8]. Load-balancing algorithms classified by Casavant to be found between dynamic and static scheduling class. This means that there can be traced recognizable features from both classes in a load-balancing algorithm. There are however little reflection or discussion around this classifications in the Web-Crawler Network literature.

It is important to recognize that consistent hashing schemes are pre-calculated. This means that Web-Crawler Agents and Agent Tasks are placed in a fixed and the same position regardless of the state of the Web-Crawler Network. The consistent schemes are in its bare form, as the one applied in the UbiCrawler, does not dynamically change the distribution if there are unfair distributions within the Web-Crawler Network. However if a Web-Crawler Agent is lost because of some external faults, the consistent hashing schemes are managing to redistribute the tasks in a very clever way. This means that even if some features are pointing towards static scheduling, there are other traits that indicate that a consistent hashing schemes are rather dynamic. The classification of the classic consistent hashing scheme presented by Boldi et. al, therefore fit into prior classifications of load-balancing, made by Casavant, as a hybrid between dynamic and static scheduling [8]. [13] [38]

Scheme improvements

The allocation function in the UbiCrawler and the approach suggested by Nasri et. al is to visit all the web-pages on a single host. This however have an rather large impact on the usage on the consistent hashing algorithm. The impact is related to the decision of distributing Agent Tasks containing hosts on the various Web-Crawler Agents. Since the consistent hashing algorithms do not have an input of how many web-pages each host contain, the host assignments may be unfair. One Web-Crawler Agent may receive ten hosts with 100000 pages each, and another may receive ten host with 1000 pages each. This example assignment, is very unfair, but may be realistic in a system consisting of a diverse types of hosts. Nasri et. al therefore suggested to apply the Power Law Distribution to avoid unfair distributions. [38] [8]

The Power Law Distribution (also often referred to as Pareto distributions or Zipfian distributions) is in the literature often utilized to model web-structures and web-behaviour [38] [4] [24]. The distribution is a heavy-tail distribution that is applied to calculate the probability of visiting a web-site with a specific size. This probability is further passed as an input to the consistent hashing that ensure that Web-Crawler Agent receives a fair

amount of large and small hosts to visit. Nasri et. al. assumes that the Web-Crawler Network that use consistent hashing algorithm, applies a breadth-first or any other search based allocation algorithm. This search is performed individually per host. If for instance a hidden web allocation technique is utilized, as explained in section 3.2, the Power Law Distribution may not be utilized in the same manner as suggested by Nasri et. al. Therefore the solution presented by Nasri et. al. may not be considered as a general solution to schedule all types of Web-Crawler Networks.

3.3.2 IPMicra

When the idea of geographical distributed Web-Crawler Networks started to emerge in the 2000s, researches started to investigate the possibilities for improving scheduling algorithms for geographical distribution. The original approach to geographical distribution of Web-Crawler Networks called UCYMicra, was therefore improved by Papapetrou et. al with a new load-balancing system. The system is called IPMicra and is a **location-aware** system, that finds the nearest Web-Crawler Agent by utilizing properties of the IP protocol. The algorithm details is described below and the pseudo-code is found in appendix B. [41]

How it works

The concrete property of the IP protocol that IPMicra utilize, is the sub-net structure that the traditional IPv4 use to logically distinguish segments of the network. IPMicra utilize information provided by Regional Internet Registrars (RIR)s to build a hierarchy of sub-nets found on the Internet. In addition IPMicra registers organization information and geographical location etc. to each of the sub-nets. Papapetrou et. al claims that this is a trivial process, that is fast and consume little memory. Further, the Web-Crawlers are placed in one of these sub-nets, and is initially assigned with all pages known to be located. [41]

Whenever a new host is added, that is not in an all ready assigned sub-net, the delegation process starts. This delegation bases itself on probing to find which Web-Crawler Agent that have least latency between the host and itself. This probing is sent in the form of an Internet Control Message Protocol (ICMP)-ping request. This request returns the RTT between the Web-Crawler Agent and the host. This is measured and compared for each Web-Crawler Agent, to find the fastest. The sub-net where the host is located is then assigned to the fastest Web-Crawler Agent. All subsequent hosts that are visited in this subnet, is then retrieved from the responsible Web-Crawler Agent. [41]

The delegation need subsequently to conduct fewer probes, when all the subnets are delegated. When a new host is added, it is automatically assigned to the Web-Crawler Agent that the sub-net is delegated to. However, there are mechanisms that constantly store RTT of each HTTP request. When the measured statistics are hitting a threshold, the delegation for the attached subnet is repeated to identify if there exists better Web-

Crawler Agent. This is according to Papapetrou et. al, the load-balancing mechanism offered by IPMicra. [41]

There are two input parameters used by the IPMicra. This is the probing threshold and load-balance statistic threshold. The probing threshold is set to 50 milliseconds in the experiments conducted by Papapetrou et. al. This value seem to provide a reasonable distribution, but are not further discussed. The statistics threshold value is not mentioned by Papapetrou et. al, and any further analysis of this algorithm is not conducted. [41]

Classifications

The IPMicra algorithm is delegating the subnets only one time, and uses several heuristics to detect if the subnet delegations are not performing well. When the delegations are detected to perform under the expected threshold, the delegations are modified and the subnet responsibilities are moved to another Web-Crawler Agent. All the modifications are performed during the run-time of the algorithm. This is by the definitions stated by Casavant, the most recognizable feature of a dynamic scheduling algorithm and can therefore also be classified as a load-balancing algorithm.

An important detail of IPMIcra, is that the delegation need to be conducted by a single component. The subnet hierarchy is created on one central component and the probing and delegation behaviour is performed by this component. The probing is performed by sending requests to the respective Web-Crawler Agent from the central component. The Web-Crawler Agent then performs the ICMP request, and send back the received RTT. According to the classifications elaborated by Casavant, these are recognizable features in a centralized algorithm. Therefore, IPMicra can be classified as a centralized load-balancing algorithm, with a single point of failure.

Chapter 4

Machine Learning and Bayesian Learning Automata (BLA)

Chapter 4 present briefly the concepts behind machine learning and the Bayesian Learning Automata (BLA) family. The general machine learning principles are first described briefly, before focusing on reinforcement learning. Next the concept of Learning Automata (LA) is briefly described, before a more comprehensive explanation of some central terms used to describe problems and environments. Chapter 4 continues further, with describing the first generation of BLA, before describing the important non-stationary MANBP. The last part of the chapter is devoted to explain the BLA-Kalman algorithm.

4.1 Machine learning

4.1.1 General concepts

In the 1950 Alan Turing proposed a test to identify if a computer is able to mimic the intelligence of a human, i.e. ‚Artificial Intelligence (AI). The computer passes the test, only if the interrogator is not able to distinguish the answer from the computer with a human answer. To be able to answer a Turing test, the computer must be able to perceive objects, manipulate objects, understand and communicate a human language, represent knowledge, have some form of automated reasoning and adapt to new circumstances. Each of these six areas are disciplines within the field AI, and have been heavily researched the last and current century. The area of particular interest to us, is the last mentioned; adapt to new circumstances. The main characteristics of solutions in the area, is that they receives some exterior input and utilize computational power to learn a pattern. It can then utilize the knowledge of this pattern to perform one or several actions. Since this branch combine computers and adaptive learning, it is named machine learning. [51] (chapter 1)

The concepts of learning a machine is rather difficult to explain without any notion of how a basic machine. In the literature a simple machine is referred to as an agent

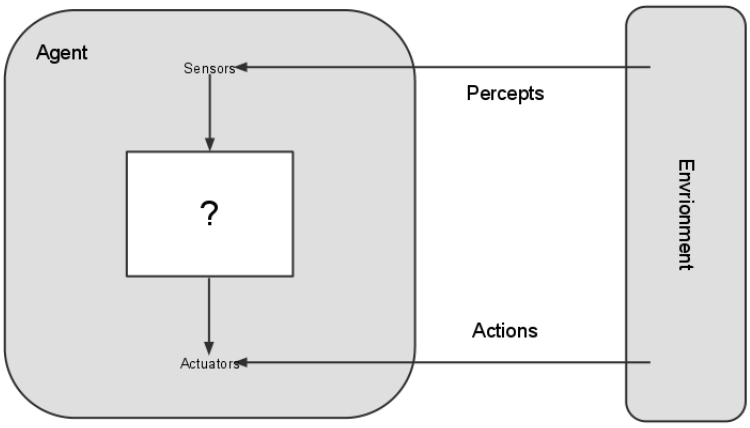


Figure 4.1: A model of an agent that interact with an environment through actuators and receive perceptions in return via sensors. [51], chapter 2

[51]. The agent communicate with an exterior environment via sensors and actuators. The sensors receives updates or perceptions from the environment, and the actuators performs some action based on knowledge registered by the sensors. The perceptions are referred by us an others as the feedback received from the environment. An agent and the described components is displayed in figure 4.1. In a real world problem an agent can for instance be a taxi driver. The exterior environment to this taxi driver agent is the roads, traffic, pedestrians, customers etc. The actuators are the steering wheel, accelerator, brake, signal etc. In addition the taxi driver need to be measured, by some kind of goal. This can be safety records, maximum profit or that the taxi driver always find the fastest route. However, how the taxi driver agent learns to achieve this goal is another problem discussed in the literature. Numerous angles of learning have been discussed. Should the taxi driver learn the traffic entirely by itself?, should the taxi driver receive knowledge from a teacher? or should the taxi driver learn by trial and error? These angles are seen as the three most important ways to learn in machine learning, and are therefore categorized as: supervised, unsupervised and reinforced learning.[51]

In a supervised learning scheme, the environment is fully exposed to the agent. The agent can see the effects of its actions and based on this, learn a specified action when a similar observation is registered later. This scheme can work in for example text phrase recognition agents, where you train the agent with pre-classified text. When the agent then later receives text that is similar to the pre-classified text, it should recognize and classify the text correctly. Contrary the unsupervised scheme has no supplied output values. The agent that is trained by an unsupervised learning scheme receives only unlabeled examples. Contrary to both the supervised and unsupervised learning scheme, the reinforced learning scheme bases itself on a reward and penalty strategy. The agent receives a penalty when performing an action that is negative to the goal it tries to achieve. However

it receives a reward when performing a 'good' action. For example a taxi driver agent, may receive a tip if driving safely. The reinforced scheme is very suitable to solving problems where you only receive simple feedback, and have no friendly teacher to learn the agent how the environment responds. This is particular interesting in web related problems, because of the little knowledge the protocols and web structures can provide. An example of a problem that is solved with reinforced learning, is the re-crawling problem solved by Granmo et. al [24]. In this problem a LA is used to determine the optimal re-crawling frequency of a particular web-page, only by receiving a penalty when the page is not updated and a reward when updated. Since the particular load-balancing problem at hand is web related, we forget the supervised and unsupervised schemes and focuses entirely on reinforced schemes. [51]

4.1.2 Reinforced learning

As explained above, in reinforced learning, the agent only receives some kind of simple and scalar feedback to learn an optimal action. The feedback received will, dependent on the environment, be a result of a previous performed actions. This means that actions may cause a change in the environment, that give a negative or positive feedback. Further this means that a feedback and action pairs may be associated. It is therefore most desirable in many reinforced scheme to find the most profitable action and feedback pairs. [39] [51] [12]

To be able to find the most profitable action and feedback pairs, it is important to try several series of actions. In addition the environment can return random or noisy feedback. This type of environment is often referred to as a stochastic environment. By communicating with a stochastic environment, it is even more important the agent repeats the actions and and constantly measuring to identify the pairs giving the best mean feedback. There are however a well known dilemma related to this identification process. The dilemma is shortly that when receiving positive feedback from a series of actions, should the agent continue performing these actions or should it pursue others. If the agent continues to exploit the current most profitable actions, others more profitable actions may be ignored. Contrary, if the agent strategy is to try out other solutions, it can explore too much and never find the most desirable action pairs. This often referred to the exploration vs. exploitation battle. [39] [51] [12]

The exploration vs. exploitation battle can be found in most problems solvable by agents receiving reinforced feedback. One of these problems are the k-armed bandit problem, where the agent, i.e. , the bandit player, tries to identify the most profitable arm. The feedback the bandit player receives, is simply a penalty when pulling a non-profitable arm, and a reward when pulling a profitable arm. To find a proper trade-off between exploration vs. exploitation in the k-armed bandit problem, is therefore the subject of research conducted by Granmo [22]. The result is an agent that based itself on prior knowledge, i.e. , Bayesian reasoning. The agent is a variant of a LA, and accordingly named Bayesian Learning Automata (BLA). The details of BLA is covered in section

4.2.1

4.1.3 Learning agents and LA definitions

A bandit player or any learning agent can be formally defined as a set of internal states Φ, a set of output actions α and a set of possible updates β. In addition the transition functions F and action functions G are defined. The standard notation is showed in the quintuple in equation 4.1 and the set definitions in equation 4.2. [12] An example of how a learning agent operates is that an input β_1 is received by the sensors from the environment. The transition and action functions are then executed, with β_1 and the current state ϕ_1 as inputs. The transition function returns, according to the logic, the next state the agent is entering. Further the action functions, are deciding the next action performed by the learning agent. These definitions and behaviours are similar to the definition and behaviour of a finite-state machine. A finite-state machine is one of the main concepts behind Automata theory. In addition the main objective for the agent is to learn and therefore the definitions are identical or have similarities to LA definitions published in the literature. [37]

$$\langle \alpha\beta, \Phi, F, G \rangle \tag{4.1}$$

$$\begin{aligned} \Phi &= \{\phi_1, \phi_2 \dots \phi_n\} \\ \beta &= \{\beta_1, \beta_2 \dots \beta_m\} \\ \alpha &= \{\alpha_1, \alpha_2 \dots \alpha_m\} \end{aligned} \tag{4.2}$$

Deterministic

$$G(\bullet, \bullet) = \phi \times \beta \rightarrow \alpha \tag{4.3}$$

$$F(\bullet, \bullet) = \phi \times \beta \rightarrow \phi \tag{4.4}$$

Stochastic

$$\alpha(n) = G[\phi(n)] \tag{4.5}$$

$$\phi(n+1) = F[\phi(n), \beta(n)] \tag{4.6}$$

There are two main types of LA mentioned in the literature, one that is deterministic and one that is stochastic [12]. The deterministic have fixed states and transition and action functions. Contrary, the stochastic LA each state output is decided by a certain probability. This is reflected in the various transition and action functions displayed in equations 4.3 - 4.6. As displayed in equation 4.5 and 4.6, the variable n is used. The variable n is defined as a time step in the logical environment the automation lives. The stochastic transitions and actions therefore varies randomly with time.[37] [12]

4.1.4 Environment definitions and terminology

There are a large amount of properties that can be determined from analysis a specific environment. These properties are important to recognize when conducting research on new techniques, or adapting techniques to work in a new environment. Many of the terms, would also have very similar defenitions to the load-balancing classifications made by Casavant. These are described in section 2. Each of the properties are relatively comprehensive subjects on their own, but we intend to introduce and explain briefly the properties presented by Russel et. al [51]. Further details we therefore refer to [51].

Fully vs. partly observable

When observing an environment from an agent perspective, it can have an open view and observe all properties of the environment. This is often referred to as a fully observable environment. A fully observable environment exposes all the aspects of the environment to the agent, and the agent does not need to keep an own state to navigate in it. However when only able to detect a few aspects of the environment, it is referred to as a partially observable. An example of a partially observable environment is the k-armed bandit problem, where only the feedback from the arm is visible to the agent.

Deterministic vs. stochastic

From an agent perspective the environment may respond in two separate ways. The environment state can either be entirely predictable when performing a specific action, or it can have a certain probability associated with the current action and state. The predictable environment is often referred to as a deterministic environment. All environments that are not deterministic are then seen as stochastic.

Episodic vs. sequential

It is crucial for an agent to know, if the action it performs is based on earlier actions. An agent that is for instance checking a car-part on an assembly line, is not dependent on any previous actions to determine if the part meets the quality standards. However in many other environments, the action the agent performs is changing the environment. The future feedback received, is then dependent on the particular actions chosen in previous steps. For instance an agent playing the game of chess, is changing the game situation each time it performs an action. The future feedback will be influence by this feedback, and the agent need to make counter actions to correct missteps. To distinguish the two environment types, Russel et. al have therefore classified them as episodic or sequential. The episodic, is the environment that do not respond feedback influenced by earlier actions.

Discrete vs. continuous

Environment types can be distinguished with how the time, state, feedback and actions are handled. The main conception of how these dimensions work, is if they can be represented as a discreet value or not. In the game of chess each move can be represented by an integer and the various states can all be described with integer Cartesian coordinates. This means that the game of chess is a discrete environment. However a taxi driver agent, can not steer the car by turning the wheel in a discrete manner. Therefore, environments that is not possible to represent with discrete steps, classified as continuous environments.

Cooperative vs. competitive

In an environment, single or multiple agents can operate. When for instance playing the game of tennis, each player can be perceived as an agent playing against each other. However, there are two notions of multi-agent problems. One that describes agents trying to beat each other in a game, and one that tries to cooperate to achieve a common goal. A typical competitive environment is the mentioned game of tennis. However, if we add two an extra player agent on both teams, the agents need to cooperate to obtain a common goal. We therefore have two multi-agent environments: cooperative and competitive

Non-stationary vs. stationary

The last classification mentioned by Russel and Norvig is related to if the environment is constantly changes by time or not. A change can for example be that the repeatedly mentioned taxi driver agent, can not select the same route each time and expect a tip for being fast. The route can suffer from congestion and other periodical problems that is hindering traffic. In more formal terms, the probabilities for receiving a tip from the customers is changing, dependent on the time the action is performed. This environment type is usually classified as as a non-stationary or a dynamic environment. Contrary, when the probabilities are not changing, it is classified as stationary or static environment.

4.2 Bayesian Learning Automata (BLA) and related problems

4.2.1 The first generation Bayesian Learning Automata (BLA)

Prior knowledge of the problem, is seen as one of the most important roles in all kinds of learning. Researches within the fields of AI, computer science, and psychology have revealed that prior knowledge is important to how an agent/person is able to learn [51]. Prior knowledge was also the subject of Granmo, when developing a new LA to solve

the Two-Armed Bandit (TABB) problem [22]. The selected technique to build this prior knowledge is Bayesian reasoning and the name is accordingly Bayesian Learning Automata (LA). Bayesian reasoning is the prior estimation technique used to weight the probabilities associated with the arm selections. The background for selecting Bayesian reasoning is stated by Granmo:

Bayesian reasoning is a probabilistic approach to inference which is of significant importance in machine learning because it allows quantitative weighting of evidence supporting alternative hypotheses, with the purpose of allowing optimal decisions to be made. Furthermore, it provides a framework for analyzing learning algorithms.[22]

$$f(x;\alpha,\beta) \quad = \quad \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\,du} \tag{4.7}$$

Informally, the Bayesian approach can be described in the following manner: The prior knowledge is utilized by the BLA to calculate the most likely optimal arm to pull, before it is pulled. The Beta probability density function, displayed in equation 4.7, and its corresponding cumulative distribution function, is performing this calculation in the BLA. The BLA stores a parameter pair (α,β) for each of the arms on the Two-Armed Bandit. Further two random values are sampled from the Beta distribution using both the stored pairs. The pair that receives the largest sampled value 'win', and the corresponding Arm is selected. When feedback is received from pulling this arm, it is registered and the α,β parameter pair is updated. For instance if $Arm1$ is selected and a penalty is received, β is increased by one for $Arm1$. Contrary, if a reward is received, α is increased by one. When increasing β the distribution get more 'heavy-tailed' towards zero, thus give a larger probability of drawing a smaller random number at next iteration. However when increasing α , the Beta distribution tend to get more 'heavy-tailed' towards 1. This increases the probability of sampling a larger value at next iteration. After a certain amount of iterations the ratio between α and β, stabilizes for both arms. The optimal arm is then hopefully found. [22]

The use of random sampling from a probability distribution is also a clever way to ensure that arms that not is considered the most profitable is explored. This is valuable when finding the trade-off between exploration and exploitation. However when need to deal with a non-stationary variant of the Two-Armed Bernoulli Bandit (TABB) problem, the BLA do not perform well. This because it needs to receive an unknown amount of feedback values to be able to adjust to the ever changing settings. [12] Therefore Granmo et. al suggested a new scheme called BLA-Kalman. This scheme is designed to adopt to the non-stationary MANBP.

4.2.2 The non-stationary Multi-Armed Normal Bandit Problem (MANBP)

The MANBP is a version of the k-armed bandit problem, where the agent is placed to chose between two or more arms attached to a bandit. The goal is to find the most prof-

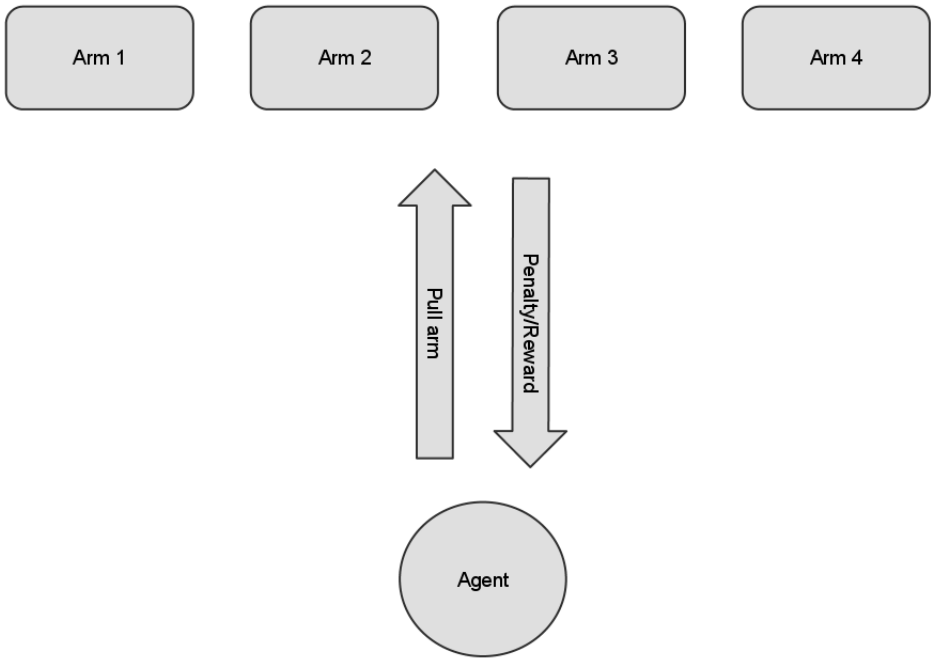


Figure 4.2: The figure displays a basic k-armed bandit and an Agent that need to find the most profitable arm. The Agent is pulling various arms, and receives a Reward if the arm selected do provide winnings. Otherwise the feedback is a Penalty.

itable arm, based on iterative selections. The k-armed bandit problem is illustrated in figure 4.2, with four arms and a single agent. The agent performs iteratively arm pulling and receives a penalty if the selected arm not provided any profit. Contrary, when pulling a profitable arm it receives a reward. To find the most profitable arm only by receiving simple feedback, is a very basic optimization problem. Therefore the k-armed problem has been the major subject for several researches when testing reinforced learning techniques. [12] [22]

The mentioned k-armed bandit environment, with only simple binary feedback and fixed reward probabilities, is a discrete stationary environment. In addition the agent does not know which arm that is the best. This means that the environment is only partly observable. However in the MANBP, the feedback scheme is rather altered and instead of receiving a simple binary feedback, the agent receives a real-number. This real-number is normal distributed, with a mean (μ) and variance (σ_{ob}). This means that the value is stochastic and the relation between an arm pull and the feedback is hidden behind Gaussian noise. Therefore we can say that the MANBP environment, in its bare form, can be classified as a partly observable, stochastic, stationary and continuous environment. [23]

The MANBP we have discussed is until now is stationary. This means that the feedback received is not changing. But since the feedback is normal distributed and stochastic, it always changes. It is therefore important to distinguish the stochastic behaviour from non-stationary behaviour. The stochastic behaviour in the MANBP environment, is the random normal distributed feedback received from the arms. This feedback have as explained earlier, a mean (μ) and variance (σ_{ob}). However when the MANBP environment behaves non-stationary, the mean and variance varies with time. This complicates the problem, because in addition to the Gaussian noise, the agent need to deal with an extra transition noise (σ_{tr}). To filter both the transition noise and the observation noise, and to find the most profitable arm was the subject of research conducted by Granmo et. al [23]. By clever usage of Sibling Kalman Filters and random Bayesian sampling, they were able to create a Automata structured agent. This agent was capable of operating in non-stationary MANBP environments.

4.2.3 Bayesian Learning Automata (BLA) with Kalman filters

As explained in sections 4.2.1 and 4.2.2, the traditional BLA handle non-stationary variants of k-armed bandit problem non-efficiently. This is solved by Granmo et. al, by using Sibling Kalman filters to filter the noise associated with the non-stationary behaviour. The output from the filters is then further used as input to a normal distribution. By continuously applying the Kalman Filters on the input, and using all prior knowledge in the process, the process can be said to be using Bayesian reasoning. In addition the agent algorithm contains individual states per arm, transitions and actions as in any LA explained in section 4.1.3. It is therefore seen as a BLA variant and referred to as a BLA-Kalman.

$$f(x_i; \mu_i, \sigma_i) \quad = \quad \alpha e^{-\frac{1}{2}(\frac{(x_i-\mu_i)}{\sigma_i^2})} \tag{4.8}$$

Kalman Filters

Kalman Filters is created to be able to predict a future state of a system despite noisy signals received from prior and current observations. The core of the Kalman Filters is clever usage of the Gaussian distribution displayed in equation 4.8 (For the sake of brevity the normalizing constant α is used). The background concepts of Kalman Filtering is complex and too comprehensive to explain in this thesis. We therefore deliberately explains a simple one dimensional example of filtering using Kalman Filters and refer to Russel et. al for more details [51].

The one step example we provide, is an one-dimensional example. This means that we look at a single continuous variable X_t. We could imagine that one-dimension is not sufficient if X_t is a position on a radar screen used for flight surveillance, but in any calculations performed by the BLA, one-dimension is enough. The basic problem is that in addition to the variable X_t, that we receive a noisy signal Z_t. We further need

to predict the X_t at the next time-step from this signal. The variables we use within the discrete time-step is the prior x_0, the observed z_1, and the predicted x_1 at time-step 1. In addition there is a sampling noise σ_0^2 associated with the prior distribution, transition noise σ_{tr} and observation noise σ_{ob}. The prior distribution is displayed in equation 4.9.

$$P(x_0) = \alpha e^{-\frac{1}{2}(\frac{(x_0-\mu_0)^2}{\sigma_0^2})} \tag{4.9}$$

To be able to track the uncertainly from one step to the next, a transition model is utilized. In the non-stationary MANBP, this is the continuous changes of the mean and variance as explained in section 4.2.2. The transition model include the transitional noise σ_{tr} and the Gaussian distribution is displayed in equation 4.10.

$$P(x_{t+1}|x_t) = \alpha e^{-\frac{1}{2}(\frac{(x_{t+1}-x_t)^2}{\sigma_{tr}^2})} \tag{4.10}$$

The observed z_1 signal, is also associated with a degree of noise. In the MANBP this noise is generated, by the variance associated with the normal/Gaussian distributed feedback. This noise is referred to as sensor noise or observation noise σ_{ob}. The related sensor model is showed in equation 4.11.

$$P(z_t|x_t) = \alpha e^{-\frac{1}{2}(\frac{(z_t-x_t)^2}{\sigma_{ob}^2})} \tag{4.11}$$

The goal when receiving the update z_1 at step 1, is to predict where the real x_1 is. This is a typical case of conditional probability $P(x_i|z_1)$. The conditional equation is defined in equation 4.15.

$$P(x_1|z_1) = \alpha P(z_1|x_1)P(x_1) \tag{4.12}$$

Equation 4.15 shows however several unknown probabilities that need to be elaborated. The value $P(x_1)$ is calculated as follows:

$$\begin{aligned} P(x_1) &= \int_{-\infty}^{\infty} P(x_1|x_0)P(x_0)dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{(x_1-x_0)^2}{\sigma_{tr}^2})} e^{-\frac{1}{2}(\frac{(x_0-\mu_0)^2}{\sigma_0^2})} dx_0 \\ &= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}(\frac{(x_1-x_0)^2\sigma_0^2+\sigma_{tr}^2(x_0-\mu_0)^2}{\sigma_{tr}^2\sigma_0^2})} dx_0 \end{aligned} \tag{4.13}$$

Equation 4.13 shows a rater complex integral, but Russel et. al show that by utilizing a simple trick called completing the square, they are able to separate the integral in two parts. One part that is an residual term, that can be placed outside the integral, and one integral of a Gaussian curve. This integral is intuitively 1, since the integral is spanning the whole Gaussian curve. After the simplification we therefore obtain the equation:

$$P(x_1) = \alpha e^{-\frac{1}{2}(\frac{(x_1-\mu_0)^2}{\sigma_0^2})} \tag{4.14}$$

It is now possible continue on the conditional $P(x_1|z_1)$ that we defined in equation 4.15 and substitute $P(z_1|x_1)$ and $P(x_1)$ with the Gaussian equations 4.11 and 4.14. The result is displayed as follows:

$$P(x_1|z_1) = \alpha e^{-\frac{1}{2}(\frac{(z_1-x_1)^2}{\sigma_{ob}^2})} e^{-\frac{1}{2}(\frac{(x_1-\mu_0)^2}{\sigma_0^2})} \tag{4.15}$$

Further Russel et. al combine the exponents and applies the technique of completing the square. The obtained function is as follows:

$$P(x_1|z_1) = \alpha e^{-\frac{1}{2}(\frac{(x_1-\frac{(\sigma_0^2+\sigma_{tr}^2)*z_1+\sigma_{ob}^2*\mu_0}{\sigma_0^2+\sigma_{tr}^2+\sigma_{ob}^2})^2}{\frac{(\sigma_0^2+\sigma_{tr}^2)\sigma_{ob}^2}{\sigma_0^2+\sigma_{tr}^2+\sigma_{ob}^2}})} \tag{4.16}$$

The Gaussian distribution displayed in equation 4.16 is rather difficult to follow. But the Gaussian function is now in its bare form and it is rater trivial to see, that σ_{t+1} and μ_{t+1} can be deduced as follows:

$$\begin{aligned} \mu_{t+1} &= \frac{(\sigma_t^2+\sigma_{tr}^2)*z_t+\sigma_{ob}^2*\mu_t}{\sigma_t^2+\sigma_{tr}^2+\sigma_{ob}^2} \\ \sigma_{t+1} &= \frac{(\sigma_t^2+\sigma_{tr}^2)\sigma_{ob}^2}{\sigma_t^2+\sigma_{tr}^2+\sigma_{ob}^2} \end{aligned} \tag{4.17}$$

There are several properties by filtering using the equations (4.17). One is that the input parameters σ_{tr}^2 and σ_{ob}^2 will adjust the calculations of μ_{t+1} in various ways. If σ_{ob}^2 is large, it pay more attention to the old mean μ_t. Contrary if the σ_{tr}^2 is large, the observed value will have more impact. This means that if the observarion is highly unreliable, the old mean would be the main basis for the prediction. However when the process is relatively unreliable, the observation is emphasised. The various input parameter properties are therefore important to remember, when selecting input values.

The BLA-Kalman algorithm

The BLA-Kalman algorithm can be viewed in algorithm 1. The algorithm is directly utilizing the Kalman equations (4.17) in an iterative manner. The algorithm is described as follows: First, $\mu[N]$ an $\sigma[N]$ is set to an initial value for all arms in the MANBP. Further the iteration process starts . Each iteration starts at step 2 with sampling a value for each arm registered. The sampled value is normal distributed and with the mean $\mu[N]$ and variance $\sigma[N]^2$ associated to the arm. In step 3 the algorithm compare all the sampled values, and select the arm that have the largest value. The selected arm is then pulled in in step 4 and the feedback is received. [23]

Algorithm 1 The BLA-Kalman algorithm

The BLA-Kalman algorithm

Input: Number of bandit arms q; Observation noise σ_{ob}^2 ; Transition noise σ_{tr}^2

Initialization: $\mu_0[N] = \mu_1[N] \ldots \mu_n[N] = A$; $\sigma_0[N]^2 = \sigma_1[N]^2 \ldots = \sigma_n[N]^2$; Typically, A can be set to 0, with B being sufficiently large.

Method:

1: **for** $N = 1, 2, \ldots$ **do**
2: For each Arm $j \in \{1, ..., r\}$, draw a value x_j randomly from the associated normal distribution $f(x_i; \mu[N], \sigma[N])$
3: Pull the Arm i whose drawn value x_i is the largest one of the randomly drawn values: $i = argmax_{j \in \{1..., q\}} x_j$
4: Receive a reward r_i as a result of pulling Arm i, and update parameters as follows:
5: $\mu_i[N+1] = \frac{(\sigma_{a_j}^2[N] + \sigma_{tr}^2) * g + \sigma_{ob}^2 * \mu_{a_j}[N]}{\sigma_i^2[N] + \sigma_{tr}^2 + \sigma_{ob}^2}$
6: $\sigma_{a_j}[N+1]^2 = \frac{(\sigma_i^2[N] + \sigma_{tr}^2)\sigma_{ob}^2}{\sigma_i^2[N] + \sigma_{tr}^2 + \sigma_{ob}^2}$
7: **for all** $i \in I$ **do**
8: **if** $i! = currentSelectedArm$ **then**
9: $\mu_i^2[N+1] = \mu_i^2[N]$
10: $\sigma_i^2[N+1] = \sigma_i^2[N] + \sigma_{tr}^2$
11: **end if**
12: **end for**
13: **end for**

In the steps 5-6 the important Kalman update process is then performed. The Kalman update process is performed by the Kalman filter equations 4.17. This update process predicts the new mean ($\mu[N+1]$) and the variance ($\sigma[N+1]^2$). These are accordingly used in the next iteration ($N+1$). Further, in the steps 7-12, the other arm states that was not modified in this iteration is updated. This update, is only adding the transition noise σ_{tr} to $\sigma_a^2[N+1]$ and keeps the $\mu_a^2[N+1]$ unchanged. The reason for this update is briefly that the arms that is not selected in this iteration, need to be pulled later. So by increasing $\sigma_a^2[N+1]$ each time it is not seleced, it increases the probability of getting explored. This is therefore a technique to solve the exploration vs. exploitation battle. [23]

Chapter 5

Load-balancing a Web-Crawler Network by applying BLA-Kalman

The previous chapters have mainly been devoted to previous techniques to load-balance distributed systems and theoretical definitions of load-balancing, Web-Crawler Networks and BLA-Kalman. Since we want to combine Web-Crawler Networks and BLA-Kalman, we need to properly define the underlying problem we are going to solve. Therefore chapter 5 are entirely devoted to define the problem mathematically and how we intend to model the load-balancing problem as a MANBP.

Chapter 5 starts by defining some notation and functions related to the web-environment the KALMAN-BLAWLB is intended to interact with. Further the definitions are thoroughly and stepwise defined, before a definition suitable in a dynamic algorithm is explained. Chapter 5 continues to elaborate the similarities between the domains and a strong relation is explained between the MANBP and the load-balancing problem at hand. Lastly some classifications of the problem relations are made before the KALMAN-BLAWLB is briefly introduced.

5.1 Mathematical definitions: Load-Balancing of Web-Crawler Networks

The base of the formal definition is the relation between the NPP described in section 2.2 and the scheduling problem at hand. There are m Agent-Tasks in an un-partitioned set Φ. All the Agent-Tasks need to be distributed, i.e., to be partitioned, for processing on n Web-Crawler Agents. We therefore see each Web-Crawler Agent queue as a partition, and state that we need to partition the set of independent Agent-Tasks into n distinct queues. The notation and equations are defined in equation 5.1 and 5.3:

$$
\begin{array}{lcl}
n & = & \text{The amount of Web-Crawler Agents} \\
m & = & \text{The amount of of Agent Task} \\
a_j & = & \text{An agent} \\
\phi_i & = & \text{An Agent Task} \\
T_q & = & \text{The queue time} \\
Q_{a_j} & = & \text{The queue of Agent-Tasks found on agent } a_j \\
A & = & \text{A set of Web-Crawler Agents } \{a_1, a_2, \ldots a_n\} \\
\Phi & = & \{\phi_1, \phi_2, \ldots \phi_m\} \\
\delta_{\phi_{a_j}} & = & \text{The distance from a Web-Crawler Agent to web-page represented by the Agent-Task} \\
g(\phi_i, a_j, \delta_{\phi_{a_j}})) & = & \text{Returns the RTT for an Agent-Task}
\end{array} \tag{5.1}
$$

$$\beta(\phi_i, a_j,) \quad = \quad T_q + g(\phi_i, a_j, \delta_{\phi_{a_j}})) \tag{5.2}$$

$$\sum_{\phi \in Q_{a_0}} \beta(\phi) \quad = \quad \sum_{\phi \in Q_{a_1}} \beta(\phi) \quad = \quad \ldots \quad = \quad \sum_{\phi \in Q_{a_n}} \beta(\phi) \tag{5.3}$$

5.1.1 Environment related functions

As explained in section 2.2, the NPP have an objective to equalize the sum of the partitions. To obtain the same goal we use the sum function, as displayed in equation 5.3, and sum the content of each Web-Crawler Agent queue. This usage introduces a problem since a Agent-Task is not a number, so just by applying a sum function to a Web-Crawler Agent do not make any sense. Therefore we need to make multiple functions that take Agent-Task as an argument and outputs a real number we can use in the sum function. These functions is the main connection between the solution we intend to present and the web-environment we intend to test it against.

The response time

A function used to return values to the sum function should return a value related to the goal(s) of the load-balancing. In a Web-Crawler Network consisting of Web-Crawler Agents, the goal is to have a fair distribution of throughput or load. Load is in the literature often described as the processing rate of the particular processor, i.e., the Web-Crawler Agent [44]. Processing rate in our case is how many Agent-Tasks that is downloaded and processed per second on that particular agent. However a processing rate per second function may return very small real number when the response time on each job is relatively high. If we consider a large Web-Crawler Network that have approximately an average of 1500 ms delay on each Agent-Task and by some point in time is 1000 Agent-Tasks in the queue. The average processing rate , i.e. , if we assume the Agent-Tasks to be processed

synchronously, would be $6.67 * 10^4$. If we think the Web-Crawler Network is further extended with more Agent-Tasks, this value will be even lower. Therefore, to make a more friendly value to measure, the $\beta(\phi_i, a_j,)$ function is defined to represent the processing time of the Agent-Task processed at a specific Web-Crawler Agent. This function is displayed in equation 5.2 and consist of the time the Agent-Task spent in the queue T_q and the RTT for the Agent-Task request.

Round Trip Time (RTT) function

The RTT is defined to include processing delay from downloading and post-processing an Agent-Task. The RTT is in addition dependent on what Web-Crawler Agent that processes the Agent-Task, since the placement and the current state of the Web-Crawler Agent may influence how fast an Agent-Task is downloaded and processed. However exact how this function is defined is dependent on the setup of the simulation environment. The function $g(\phi_i, a_j, \delta_{a_j})$ is therefore defined to return the RTT for an Agent-Task request, but the exact specification is later defined by the simulation environment.

Adding distance

As mentioned in the research questions, we intend to test our solution a geographically distributed Web-Crawler Network. By distributing Web-Crawler Agents geographically introduces a new dimension that distinguish each Agent-Task in the load-balancing problem. The dimension introduced is the distance between the assigned Web-Crawler Agent and the Agent-Task that is processed by it. This means that the distance from the selected Web-Crawler Agent to the host providing the web-pages should be as short as possible. The distance is however not in the sense of traditional geographical distance that is used to measure distances between cities etc. By distance we mean the network-topological distance that causes extra delay, i.e. , packet latency, between the Web-Crawler Agent and the Agent-Task. This means that if a Web-Crawler Agent is located in Sweden, it is not necessarily the closest Web-Crawler Agent to all Swedish web-pages. This distance is however dependent on network connections and backbones used by both the target host, and the Web-Crawler Agent. We define this distance as the function $\delta_{\phi_{a_j}}$ (equation 5.4). The function receives the selected Web-Crawler Agent and the current processing Agent-Task as an input.

The distance should be considered when each Agent-Task, i.e. , representing a web-page, should be downloaded. It is convenient to assume that it is mainly the RTT on each request that is influenced by the distance and therefore the RTT function $g(\phi_i, a_j, \delta_{\phi_{a_j}})$ must pass the distance as a parameter. However since the distance and RTT clearly is dependent on the environment the Web-Crawler Network exits in, the usage and how the calculation is performed is defined when specifying the simulation environment.

$$\delta(a_j, \phi)_{\phi_j} \quad = \quad \text{The RTT value for a particular Agent-Task} \tag{5.4}$$

5.1.2 Non-stochastic problem

By including the processing time function displayed in equation 5.2, we can utilize the sum function on the distinct partitions. This function now receives a set of outputs from the process time function for all Agent-Tasks in the partition. According to the NPP definition, discussed in section 2.2, the sum of the content for each partition should be equalized. What this essentially means for the scheduling problem at hand, is that the processing time β, should be equalized on the agents. Further load can be said to be fairly distributed if the sum is equalized. This conform to the objective mentioned in the research questions in section 1.3, that the fairness needs to be studied for the BLA-Kalman scheduling process. The partition problem at hand can therefore be defined more precisely to partition Φ into n partitions such that the sum of the processing time of each partition are equalized. Equation 5.3 shows this definition mathematically.

5.1.3 Stochastic definitions

In a simulated web environment the RTT from each Agent-Task, i.e. , the value returned from $g(\phi_i, a_j, \delta_{a_j})$, is rather rather noisy. The reason for this is the many factors that come to play for each request. The host web-server load and network traffic are two examples of factors that make the RTT for each request very noisy. However if we apply the Central Limit Theorem (CLT) on a large enough set of measured RTT values, the theorem explains that the set is normal distributed with a mean and variance [50]. We therefore use random sampling from a Gaussian distribution as a RTT function. This function is displayed in equation 5.5.

$\sigma^2(\phi, a_j) \quad = \quad$ The variance for a request to the web-page ϕ_i from Web-Crawler Agent a_j

$$X_{\phi_i} \sim \mathcal{N}(g(\phi_i, a_j, \delta_{a_j}), \sigma^2(\phi, a_j)) \tag{5.5}$$

In a relatively noisy Internet, the mean μ of the measured RTTs can vary from which Web-Crawler Agent that execute the request. This mean μ can for example vary with the distance from the executing Web-Crawler Agent to the web-server hosting the web-page. To calculate a mean for the normal distributed sampling function 5.5, the RTT function $g(\phi_i, a_j, \delta_{a_j})$ is therefore used.

The variance is assumed to be individual for each Agent-Task and dependent on which Web-Crawler Agent that executes the Agent-Task. This assumption is based on that for instance network noise may vary between Web-Crawler Agents and a specific web-page. Noise can for example be caused by a hidden process residing on the web-server that is executed for just one particular web-page. In addition the noise may be caused by transmission errors or similar issues residing on a single Web-Crawler Agent. This vari-

ance function, i.e. , $\sigma^2(\phi, a_j)$ in equation 5.5, is therefore specified to be a result of each Agent-Task and the selected Web-Crawler Agent.

Since the RTT is sampled randomly from a normal distributed variable, the earlier defined functions that utilize RTT need to be redefined. The $\beta(\phi_i)$ function displayed in equation 5.2 is after replacing $g(\phi_i, a_j, \delta_{a_j})$ with X_{ϕ_i} a stochastic function $B(\phi_i)$. This means that the processing time is stochastic determined. This implicates that the objective of equalizing the sum of each agent queue is stochastic. The new objective is a direct alternation of equation 5.3 and is displayed in equation 5.6.

$$\sum_{\phi_i \in Q_{a_0}} B(\phi_i) \quad = \quad \sum_{\phi_i \in Q_{a_1}} B(\phi_i) \quad = \quad \ldots \quad = \quad \sum_{\phi_i \in Q_{a_n}} B(\phi_i) \tag{5.6}$$

5.1.4 Stochastic definitions in a dynamic algorithm

The stochastic definition, presented in equation 5.6, introduces a few practical and logical problems when you need to measure how equal the content of each queue is in a dynamic algorithm. The definition may be solvable in a static algorithm, where it is for example possible to try several combinations and validate the equalization iteratively. This however, is quite difficult when the RTT is stochastic. An example how to solve it, is to gather a great deal of statistics and base the decisions on prior knowledge. Since we do not want to try a static approach to solve the problem, it is necessary to try to find a measurable definition that can be suitable to our dynamic approach.

In a dynamic approach, as the one we propose, the solution does not have any prior knowledge of the system. It gradually adapts by time to load-balance the system, by performing testing with several permutations Agent-Tasks. Lets say that each Web-Crawler Agent process the first job in the queue, i.e. , the job with index 1 in the queue, at time t. Since the process time for each Agent-Task is not prior known and stochastic, the first job may take longer time at Web-Crawler Agent1 than Web-Crawler Agent2 with a certain probability. This means that Web-Crawler Agent1, with the current assignment of Agent-Tasks, may need to process a job that takes longer time than Web-Crawler Agent2. At time t this is unfair and means that at $t + 1$ Web-Crawler Agent2 is probably able to start a new job while Web-Crawler Agent1 is working with the first job. This means that the Agent-Task on Web-Crawler Agent1 should maybe be transferred to another queue that is able to process the Agent-Task faster. The definition of the problem presented in equation 5.7 is therefore altered to equalize the stochastic $B(\phi_i)$ values at the specific time t.

$$N \quad = \quad \text{The maximum time-line of the Web-Crawler Network}$$

$$\forall t <= N\{B(next(\phi_i, Q_{a_0}, t)) \quad = \quad B(next(\phi_i, Q_{a_1}, t)) \quad = \quad \ldots \quad = \quad B(next(\phi_i, Q_{a_n}, t))\} \tag{5.7}$$

The definition presented in equation 5.7 would be used as a perquisite to the solution and is how we perceive the problem practically. It is however important to remember that we also include the time each Agent-Task T_q have to wait in the queue in the function $B(\phi_i)$. This means that we measure the impact on assigning the measured Agent-Task to the queue. If we leave the T_q parameter out of the equation, we only try to find the best Web-Crawler Agent that can process the Agent-Task isolated. This means that, if we skip this T_q, the problem perception ignore the combination of Agent-Tasks . This combination is obviously a central part of the combinatorial scheduling problem described earlier in section 5.1. We therefore utilize the definition presented in equation 5.7 by emphasizing the need for measuring T_q for each Agent-Task, to actually solve the scheduling problem at hand.

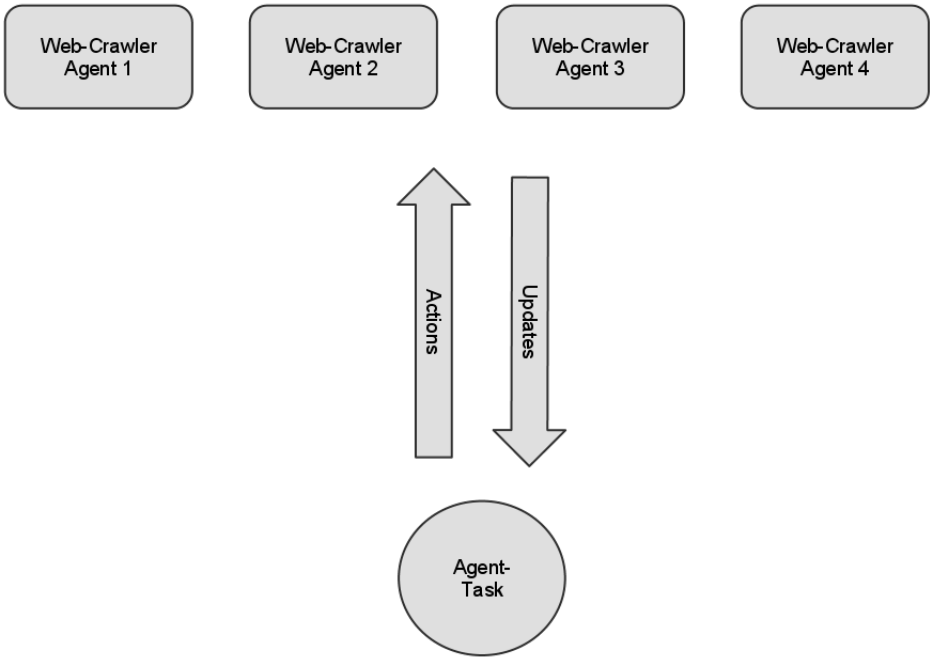


Figure 5.1: The figure shows the Web-Crawler Network domain specific MANBP interpretation. Each Agent-Task is seen as the Agent that contains the BLA-Kalman variables. The actions are seen as to choose what Web-Crawler Agent that should retrieve the webpage associated with the Agent-Task. The updates sent to the BLA-Kalman algorithm is the process time, $B(\phi)$

5.2 Combining the BLA-Kalman and Web-Crawler Network domains

Since the domains of the BLA-Kalman and Web-Crawler Networks will be combined in this thesis, we do need to compare the problems solved by the BLA-Kalman and the load-balancing problem at hand. We therefore provide, in the following sections, an explanation of how we intend to interpret and back up the combinations of the two domains.

5.2.1 Load-Balancing of Web-Crawler Networks and the Multi-Armed Normal Bandit Problem (MANBP)

As explained in section 4.2, the MANBP is used to test and verify the BLA-Kalman by Granmo and Berg [23]. Both problems are displayed in figure 5.1 and 4.2 in a similar fashion. The Agent-Task is interpreted as the Agent component that receives feedback and performs actions. The Web-Crawler Agents are perceived as the Arms that the Agent-Task need to choose between. Further explanations of the interaction between the components and why the component arrangements are selected, is elaborated in the following sections.

Agent-Task

The MANBP is as described in section 4.2 to pull Arms on a multi-armed bandit and receive feedback based on the selection. The object that performs this selection is the Agent. When defining an Agent in the load-balancing problem at hand, it is convenient to involve the stochastic problem definition in section 5.1.4. This problem definition clearly puts the Agent-Task as the object that needs to be transferred between the queues. In addition it is stated that we need to handle them individually, since we need to gradually adapt to a solution by time and not test combinations of Agent-Tasks iteratively. The Figure 5.1 therefore shows the Agent-Task as the Agent in the Web-Crawler Network specific MANBP.

The perception that places the Agent-Task as the Agent was not the only configurations that was considered. One configuration alternative that was considered, was to perceive the Web-Crawler component; Scheduler as the Agent. The Scheduler is as explained in section 3, a component that have the responsibility of distributing Agent-Tasks. It is only one such component in a Web-Crawler Network, and compared with the selected configuration the Agent need to be centralized and contain one state. If it is possible to avoid a centralized algorithm, as described in section 2.1.3, we keep away from storing too many global and centralized states and therefore abandoned this configuration.

As described in section 3.3.1 the Consistent Hashing scheduling algorithm utilize a host-based representation of each Agent-Task. This means that an Agent-Task is attached to a host, and when assigned to the Web-Crawler Agent it is telling it to visit a single or

multiple web-pages on this web-host. The exact web-page that the Web-Crawler Agent need to retrieve, must then be decided by an externally defined allocation scheme. However, an Agent-Task can also be representing a web-page, i.e. , sometimes referred to as URL-based scheme. The advantage of using an URL-based scheme in our solution, is the fact that each unique web-page have own cost values associated with it [4]. This cost values can be a associated with for example individual processing time, byte sizes or other types of communication delay. This means that the RTT is individually determined, and as explained in section 5.1.3, have its associated mean and variance. By using an URL-based scheme we therefore can load-balance based on individual, and possible large differences between web-pages on the same host.

On the other hand, an URL-based scheme immediately creates a problem if we think scalability. Since a single host can contain thousands or even millions of web-pages. This causes a scalability issue and a little predictable scenario if we add for instance 100 hosts with a varying amount of attached web-pages . A host-based scheme would clearly be more scalable and would at the same time have an associated mean and variance. The mean and variance would however be measured for all the web-pages selected by the allocation scheme on the particular host. This hides the individual differences between the web-pages, and the decision the BLA-Kalman could be less accurate than in an URL-based scheme. However the difference is not considered that significant and we want our solution to be scalable and have a predictable amount of Agent-Tasks. We therefore consider each Agent-Task to represents a host, e.g. , vg.no, integrasco.no etc, and further suggest the prototype to use a host-based scheme.

Web-Crawler Agent

The Agent in the **MANB!** (**MANB!**) need to be perform updates and get feedback from an multi-armed bandit. Since the problem definitions elaborated in section 5 see the problem as to distribute Agent-Tasks on multiple Web-Crawler Agents, makes it convenient to treat each Web-Crawler Agent as an Arm. The definition of pulling and Arm is to assign an Agent-Task to the Web-Crawler Agent queue. Therefore the Multi-armed Bandit is seen as an unified component of all the Web-Crawler Agents, and each Web-Crawler Agent as an Arm. This is displayed in figure 5.1

The Updates and Actions

We have now defined Agent-Tasks, and the role of the Web-Crawler Agent in figure 5.1. This leave us with defining the updates and action, i.e. , sometimes referred to as the feedback scheme. The update values received is used by the BLA-Kalman to decide which Web-Crawler Agent that should process the Agent-Task. The action is defined to be exactly this decision. To be able to define properly this scheme, we therefore involve the problem definitions elaborated in section 5.1.

The definitions presented in section 5.1, clearly defines the problem to equalize the

queue content, or more specifically in section 5.1.4 to equalize the retrieval continuously across the queues. The equalization goal should be perceived and used as an objective when defining the update. The first alternative, that is relevant to the goal, would be to use for example a deviation parameter like the standard deviation between the stochastic retrieved response values ($B(\phi, a_j)$). By utilize the BLA-Kalman to minimize the standard deviation value and use a 'less-is-better' reward scheme would probably be an applicable solution. However, as explained in section 5.1.4, we deal with a dynamic problem with a timeline. If we utilize a technique like the standard deviation in a such problem environment, we need to measure the standard deviation on each time-step t. This need to be queried in the Web-Crawler Network continuously when each Agent-Task is processed. This type of scheme could cause undesirable much communication overhead in the Web-Crawler Network. It is therefore more appropriate to use another more less global value as an update value to the Agent-Task.

Since global measurement variables are more or less out of the question, the update scheme needs to receive a value that can be queried within each Web-Crawler Agent. A relevant value that is constrained within the Web-Crawler Agent is the value returned from the function $B(\phi)$. This value, is as explained in section 5.1.4 and section 5.1.1, include the value T_q in addition to the stochastic RTT function displayed in equation 5.5. It is also explained in section 5.1.4 that by including the queue time T_q, we measure the impact the assignment have on the whole Web-Crawler Agent queue. If we therefore use the value returned from $B(\phi)$ as an update value and modify the BLA-Kalman algorithm to use a less-is-better reward scheme, it is probably possible to dynamically load-balance the Agent-Tasks to achieve an higher degree of fair load-balancing

5.2.2 The Non-Stationary Web-Crawler Network

As explained in section 4.2.2, the BLA-Kalman is tested to solve the non-stationary MANBP. The non-stationary MANBP is shortly that the probabilities for receiving specific updates is changing and make it harder to identify 'positive' actions. To transfer this definition into the Web-Crawler Network, it is necessary to look at both the domain interpretations we made in section 5.2.1 and the formal definitions in 5.1. The domain interpretations places the Agent-Task in the Agent position that performs Actions. The Arms are seen as the Web-Crawler Agents and the Updates are the responses from the function $B(\phi)$. The properties of the $B(\phi)$ is therefore important to acknowledge, since the non-stationary behaviour is based on probability changes between updates received from the Web-Crawler Agents.

The two components in $B(\phi)$, T_q and the stochastic X_{ϕ_i} variables, plays a central roles in the non-stationary perception as well. The T_q variable is in reality measuring the total Web-Crawler Agent load, from the decision action is performed to the update is received. If we had a stationary problem, T_q and X_{ϕ_i} would have to return a the same mean μ value each time the Agent-Task is assigned to a Web-Crawler Agent. Since this time is dependent on the other assignments, and the noise associated whit the hosts, this

value will vary with the process-time responses of the other assignments. This means that the probabilities for receiving a specific update is constantly changing, or informally that the capacity changes constantly on the Web-Crawler Agents. The MANBP itepretations , displayed in figure 5.1, can therefore be said to be non-stationary.

5.2.3 Multi-Agent classifications

The first and most interesting property of the problem, that is described in section 5.2, is that it consists of several Agent-Tasks. Each of these Agent-Tasks are perceived as the Agent that need to interact with the Web-Crawler Agents. This type of systems are often referred to as a multi-agent environments. [51].

Multi-agent environments have several properties, and one of them is if the Agents cooperate or are competitors. The difference between these are briefly that cooperative Agents try to achieve a common goal and competitive Agents play a game. When playing a game, the Agents performs actions and have strategies that the other Agents may benefit or lose game-points on. An example a typical game is chess. In the game of chess the Agents are performing several strategies and either loses or win the game. When one Agent wins the game (+1) in chess the other loses the game (-1). These types of games are often referred to as zero-sum games, deterministic games etc. The cooperate scheme, is on the other hand, coordinating its strategies to obtain a common goal. To do this, it need to enter a joint effort, have a specific plan how to obtain the goal and coordinate it. A typical joint effort is the game of double tennis, where there are two Agents cooperating to win the match. These needs to have a plan and coordinate their next move to obtain this goal. [51]

The multi-agent environment we propose, has a common goal of equalizing the load on each of the Web-Crawler Agent. But the Agent-Tasks will in many cases try to compete with each other to achieve this goal. One Agent-Task may perform an action that harms another Agent-Task, by giving it longer processing time. However the Agent-Task that performed the harmful action, could itself be punished by receiving and unfavourable update. This is possible since we suggest to use 'less-is-better' update scheme, that favour the Web-Crawler Agent that give the smallest process time $B(\phi)$. It is therefore convenient to assume that the solution we suggest consists of an array of cooperative Agents trying to assign Agent-Tasks in such way that load is equalized across the Web-Crawler Agents.

To cooperate and achieve a goal, the literature describes that a plan is needed [51]. The load-balancing approach we suggest need to also have some kind of plan. The plan could for instance be to try out several combinations of Agent-Tasks and measure the load-equalization. The problem at hand, as explained in section 5.1.3, have however a time-line. This demands continuous planning, that consists of utilizing the exploration vs. exploitation characteristics of the BLA-Kalman. This exploration vs. exploitation would, dependent on the parameters, try to learn an equalized assignment of the Agent-Tasks. Together with the 'less-is-better' update scheme this is seen as the plan to achieve

the common goal of equalizing the process time on each Web-Crawler Agent.

5.3 Solution: KALMAN-BLAWLB

The next step after defining the problem formally and combining the domains, is to approach the problem in a more practical manner. Therefore we take the explained domain elements and try to fit it into a suitable Web-Crawler Network architecture and test it in a simulated web-environment. We name our solution KALMAN-BLAWLB after the BLA-Kalman, add 'W' for Web-Crawler Networks and 'LB' to identify that it is a Load-Balancing algorithm.

Chapter 6

The KALMAN-BLAWLB

In previous chapters the problems and solutions related to scheduling and load-balancing in are elaborated. In addition the details of the BLA-Kalman and how the BLA-Kalman can be coupled with the load-balancing problem at hand, are elaborated in chapter 4 and 5. Chapter 6 is intended to explain the solution called KALMAN-BLAWLB and starts with discussing some design considerations. A high level architecture, suitable for a Web-Crawler Network is further deduced and explained as the KALMAN-BLAWLB architecture. The KALMAN-BLAWLB algorithm is further explained by elaborating the component definitions and a detailed 28 step algorithm is presented. Lastly, the implementation is very briefly explained.

6.1 Design considerations

In the next sections we intend to present a design and architecture. There are , however , some coniderations that er need to elaborate and these are presented in this section.

6.1.1 Separation of allocation and load-balancing

We intend to separate the scheduling, i.e. , the load-balancing algorithm, and the allocation algorithm in the design. The term confusions between these two are discussed in chapter 2 and indications lead to that scheduling and allocation are combined in some cases. However we see these algorithms as separable in our concrete load-balancing case and an allocation algorithm is regarded as a completely separate algorithm to our solution. Therefore whenever an allocation algorithm is referenced, it is referenced as an external algorithm. A such reference may be a call executed by a Web-Crawler Agent to retrieve a page from a specific host. The allocation algorithm is then independently deciding which web-page located on this host that is going to be downloaded.

6.1.2 Multiple instances of Web-Crawler Agents

The KALMAN-BLAWLB is intended to be duplicated in several parallel processes. In other words the Web-Crawler Agent will have several copies of itself running on several machines or processors. This type of copy is seen as a run-time copy of the algorithm, that consists of instructions that are able to run on a computer. To differentiate the run-time copy and the algorithm, we use the term instance. When we reference an instance, we mean a single run-time copy of the KALMAN-BLAWLB algorithm. These instances should be able to run in a parallel, since the problem of load-balancing Web-Crawler Networks is a problem where the load must be equalized at a specific time.

6.1.3 Blocking request

Synchronous or blocking request calls are the mode that each Agent-Task should be processed. By a blocking request we mean that each Web-Crawler Agent processes an single Agent-Task and blocks all further actions. This blocking request policy may not be the most ideal system utilization wise, when considering that each Web-Crawler Agent in a real situation could be located on a computer that is able to execute multiple HTTP requests at the same time. However for simplicity, we utilize blocking requests against the environment.

6.1.4 State independence

A Web-Crawler Network have naturally a lot of state information attached to several of the components. This would also be the case in KALMAN-BLAWLB. It is however convenient to keep this state information as little redundant as possible to avoid consistency differences between the variables. This means that an Agent-Task currently residing in Web-Crawler Agent 1, must have access to the same associated state variables when it for instance is moved to Web-Crawler Agent 2. This is important because the Agent-Tasks are the objects we want to load-balance and will be transported between the Web-Crawler Agents. We also base the load-balancing on the BLA-Kalman algorithm. By keeping the state information as close as possible to each Agent-Task, we can operate BLA-Kalman without querying the Web-Crawler Network for state information. It is therefore stated that we need to base the load-balancing on states that are independent and kept close to each Agent-Task.

6.2 KALMAN-BLAWLB architecture

The load-balancing algorithm, in figure 6.1, is designed with the main architecture, i.e. , the architecture shown in figure 3.1, of any traditional Web-Crawler Network or Web-Crawler system in mind. To simplify and decrease the amount of components, we propose

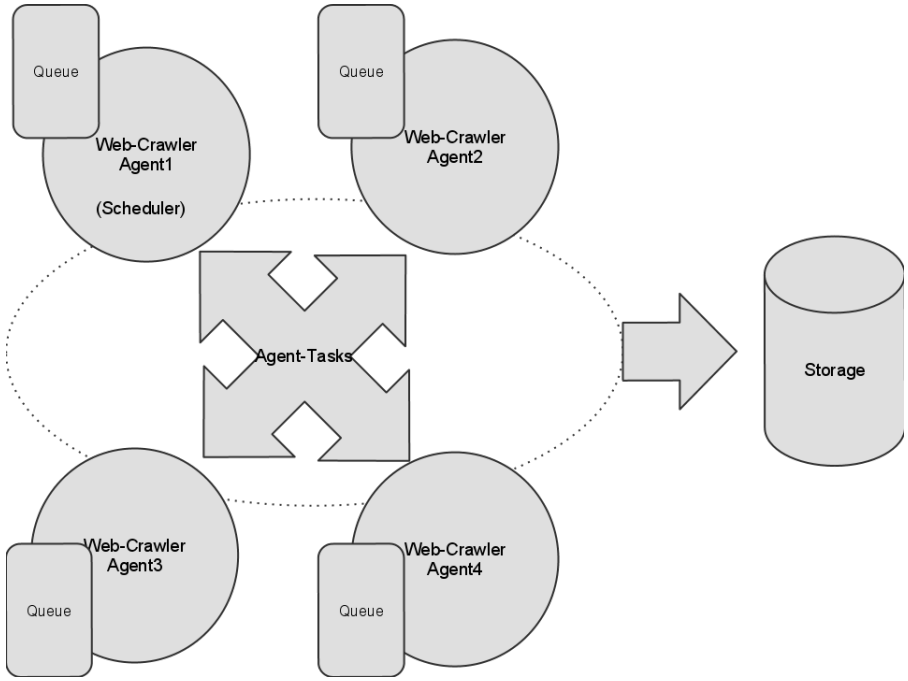


Figure 6.1: The figure shows an overview of a KALMAN-BLAWLB Web-Crawler Network architecture with four Web-Crawler Agents where each is operating a Queue and is connected to a common Storage component . Each Web-Crawler Agent is operating independently and distribute Agent-Tasks to each other. However Web-Crawler Agent1 is set to be responsible for the Scheduler role. All the Web-Crawler Agents in the Web-Crawler Network exists in a web-environment, but this is not displayed in the figure.

an architecture where the components Scheduler and Multi-threaded downloaders are replaced with a general single component called a Web-Crawler Agent. The described ring architecture is common in distributed systems and described by Tanenbaum et. al. in the well known book Distributed Systems: Principles and Paradigms [55] (page 369).

The architecture shown in figure 6.1, displays Web-Crawler Agent1 as the Scheduler. Web-Crawler Agent would for instance be in charge of distributing the Agent-Tasks initially to the other Web-Crawler Agents placed in the Network. The Agent-Tasks are executed on the assigned Web-Crawler Agents and the resulting data is sent to the Storage unit. The Agent-Task may then be transported back to the Scheduler, or to another Web-Crawler Agent in the Network. This implicates that Agent-Tasks can be sent and received between all the Web-Crawler Agents, making the architecture flexible. However if the Web-Crawler Agent that have the role as Scheduler fails, no responsible component is present take over. A realistic example could for instance be that the communication break down between the Scheduler instance and the other Web-Crawler Network. The time an instance holds the Scheduler title should therefore be kept at a minimum and considered by the Web-Crawler Agent in the specification of the KALMAN-BLAWLB algorithm.

The architecture presented in figure 3.1, have no particular requirement to where the

Web-Crawler Agents need to be placed. The Web-Crawler Agents can then be placed in for example separate countries or even separate continents. The architecture presented have in addition clear similarities, with the geographical distributed Web-Crawler Agents presented by Papapetrou et. al. [40]. However the architecture, gives the operator the opportunity to place all the Web-Crawler Agents at the same location. Therefore the architecture can be said to be an optional geographical distributed architecture.

The architecture displayed in figure 3.1 is, in addition to be flexible, rather easy to either extend or reduce . The figure shows four instances of Web-Crawler Agents, but this amount could be extended further in a scale-out fashion. As explained in section 1.1.1, this is common way to scale Web-Crawler Networks. However the scalability is dependent on the size and capacity of the second component in the architecture; the Storage component. The Storage component, as it is displayed in figure 3.1, is probably not scalable. But since we do not intend to store anything in the proof of concept solution portrayed in this thesis, we consider the architecture presented scalable.

6.3 KALMAN-BLAWLB component definitions and algorithm

The KALMAN-BLAWLB algorithm is designed as with the architecture described in section 6.2 in mind. The architecture describes one component, i.e. , the Web-Crawler Agent, that have several responsibilities in the Web-Crawler Network. The Web-Crawler Agent process the Agent-Tasks by querying the web-environment. Further it needs to act as an Scheduler, since this is not an own component in the Web-Crawler Network architecture suggested. This implicates that the Web-Crawler Agent need to decide where the Agent-Tasks should be processed next and state information must be shared. To further explain the details of how we intend to solve the roles and incorporate load-balancing in KALMAN-BLAWLB, we first present some definitions and symbols. Second, the algorithm is presented as a step procedure and explained thoroughly.

6.3.1 Component definitions

A complex system like a Web-Crawler Network, have several subordinated components related to concrete operation of the algorithms involved in the system. This is sometimes referred to as the domain definitions of the algorithm, and should by our standards be heavily related to the formal definitions of the load-balancing problem at hand. The formal definitions are presented in section 5.1. The modified and new domain related components are therefore introduced and further explained.

Agent-Task

As described in the formal definitions in section 5.1, each Agent-Task is defined as ϕ. However what each Agent-Task represent in a Web-Crawler Network, must be defined and is algorithm specific. In many cases an Agent-Task may only need to represent a host or an URL. However, the solution we propose, want to combine BLA-Kalman state information and use a host based scheduling scheme as discussed in section 5. In addition it is described in section 6.1 that we need to keep the state information as close as possible to each Agent-Task, to avoid state redundancy etc. It is therefore convenient to define each Agent-Task ϕ as an object containing the state information and other variables that is independent to each Agent-Task.

$$\begin{array}{lcl} N & = & \text{A discrete feedback counter} \\ T_q & = & \text{The queue time-stamp} \\ l & = & \text{The indentificator of the web-page that should be downloaded} \\ \mu_a[N] & = & \text{The mean queue process time estimate for Web-Crawler Agent } a \text{ on update } N \\ \sigma_0[N] & = & \text{‘ The standard deviation associated Web-Crawler Agent } a \text{ on update } N \\ \phi & = & (\{\mu_0[N] \dots \mu_n[N]\}, \{\sigma_0[N] \dots \sigma_n[N]\}, \sigma^2_{ob}, \sigma^2_{tr}, T_q, l) \end{array} \tag{6.1}$$

The object definition of ϕ , displayed in equation 6.1, contain the mean queue variables $\mu_a[N]$ together with the associated standard deviation $\sigma_a[N]$ values. These values represent the current BLA-Kalman state-information, when N updates are performed on each Web-Crawler Agent. In addition the predefined BLA-Kalman observation noise σ^2_{ob} and transition noise σ^2_{tr} are present in the object.

The BLA-Kalman variables are earlier explained in section 4.2. However the additional variables T_q and l are not mentioned yet. The variable T_q is a discrete time value that is set to the current time, i.e. , by using the method $now()$, each time the Agent-Task is entering a queue. This value is further utilized in the $B(\phi, a_j)$ function as explained in section 5.

The second new variable we introduce is the variable l. This variable contain the exact location to the web-page that is to be retrieved. The format may be in a form of an URL or an integer. The exact format is depended on if the web-environment is real or simulated. A simulated web-environment, can for instance be simulated by using an integer indexed key/value data-structure. Since all further tests will be performed in a simulated environment, the variable l is considered to be an integer.

Queues and pointers

$$
\begin{array}{lcl}
n & = & \text{Number of Web-Crawler Agents} \\
m & = & \text{Number of Agent-Tasks} \\
a_j & = & \text{Current agent (equivalent to this/self etc)} \\
a_r & = & \text{A selected agent} \\
Q_{a_j} & = & \text{An First In First Out (FIFO) queue of Agent-Tasks found on agent} \\
A & = & \text{A set of registered agents}
\end{array}
\tag{6.2}
$$

As described in the formal definitions in section 5.1, the scheduling task at hand is to distribute Agent-Tasks to several Web-Crawler Agent queues. The queues are shown in figure 6.1 and are represented by Q_{a_j}. This queue is in our approach a FIFO queue. Whenever an Agent-Task ϕ is added to the FIFO queue Q_{a_j}, it is appended to the beginning of the list. The Agent-Tasks are further removed one by one from the end of the queue, when the it is iterated. In addition the queue is supporting concurrent operations, allowing it to receive Agent-Tasks in a transactional manner. The queue Q_{a_j} is therefore a thread-safe FIFO, that can be read and written to by any Web-Crawler Agent in the Web-Crawler Network.

The variable a_j is utilized in further definitions as a *self* or *this* pointer to the current Web-Crawler Agent. A *this* or *self* pointer is regularly seen in object oriented programming languages like Java and Python, to reference the current object. An example is if the Web-Crawler Agent with the unique id 1 is using the variable a_j as a parameter in a function. The function then receive the integer id 1 as an argument, and can for instance un-register the agent from a register or similar. The parameter a_j can therefore be seen as a current agent pointer, represented as an unique agent identification.

Similarly to the self pointing a_j, the variable a_r is pointing towards a selected Web-Crawler Agent. The variable a_r must be found present in the set A and is defined to be a non-zero integer.

6.3.2 The KALMAN-BLAWLB algorithm

An algorithm that need to operate in a Web-Crawler Network, must naturally implement communication routines, involve concurrency etc. This is by us, handed over to the implementation and the framework. We fully concentrate on how the interactions between the environment is and how the BLA-Kalman learning is integrated in the domain of the Web-Crawler Agent and Web-Crawler Network generally. The KALMAN-BLAWLB is therefore presented as a 28 step method in algorithm 2. In addition we present the update and decision scheme as own independent methods in algorithm 3 and 4.

Algorithm 2 The KALMAN-BLAWLB algorithm

Input: initialScheduler,n, m, InitMu, InitSigma, ObservationNoise, TransitionNoise
Initialization: initialScheduler := false, $\Phi = \{\phi_n \dots \phi_m\}$ each ϕ is initialized $(\{\mu_0[N] \dots \mu_n[N] = InitMu\}, \{\sigma_0[N] \dots \sigma_n[N] = InitSigma\}, \sigma^2_{ob} = ObservationNoise, \sigma^2_{tr} = TransitionNoise, T_q = 0.0)$; a_j unique agent id between $0 \dots n$; N = 0
Method: runWebCrawlerAgent()

1: $A := \{\emptyset\}$
2: **if** $initialScheduler = true$ **then**
3: $A \cup a_j$
4: $A \cup waitForAgents(numberOfAgents)$
5: **for all** $\phi_i \in \Phi$ **do**
6: $a_r := decideNewAgentForAgentTask(\phi, A)$
7: $addAgentTaskToQueueOrSendToNew(Q_{a_j}, a_r, a_j)$
8: **end for**
9: $sendPointersToAllAgentsAndStart(A)$
10: **else**
11: $A \cup registerAgentInNetworkAndWait(a_j)$
12: **end if**
13: $s := true$
14: **while** $s = true$ **do**
15: **if** $size(Q_{a_j}) = 0$ **then**
16: $q := waitAndRetrieveAgentTasks(a_j)$
17: $Q_{a_j} \cup q$
18: **end if**
19: **for all** $\phi \in Q_{a_j}$ **and** $s = true$ **do**
20: $retriveAndUpdateNewAllocation(\phi)$
21: $visitWebPage(\phi, a_j)$
22: $g := (now() - T_{q_\phi})$
23: $updateBLAWithResponse(\phi, g, a_j)$
24: $a_r := decideNewAgentForAgentTask(\phi, A)$
25: $resetTimeTQOnAgentTask(a_r)$
26: $addAgentTaskToQueueOrSendToNew(Q_{a_j}, a_r, a_j)$
27: **end for**
28: **end while**

An brief overview

The algorithm can briefly be described by four significant parts. The input/initialization values, the initialization phase (step 1-12), operational phase (step 13-28) and decision phase (step 23-24). The input and initialization values decide the role of the Web-Crawler Agent, i.e. , if it should be a Scheduler or not, and the BLA-specific parameters. In

addition the Agent-Task objects, i.e. , described in section 6.3.1, are initialized with a common set of initialization values. The initialization phase is performed by all Web-Crawler Agents, but the Web-Crawler Agent with the Scheduler role must in addition to initiate itself, initiate the Web-Crawler Network. This initialization is simple, and is shortly to perform the initial distribution of the Agent-Tasks, while the other Web-Crawler Agents are in a blocking wait state (step 9). The operation phase is the part of the process where the crawler specific operations are executed. The Agent-Tasks are retrieved from the queue and the allocation algorithm is deciding which web-page the Web-Crawler Agent need to visit on the specific host. When the result is retrieved, the RTT is measured and the queue time is added. This is further passed as input to the BLA-decision and selection methods. This particular phase is seen as an own phase therefore named the decision phase.

The input/initialization values

The method presented in algorithm 2 is intended to have two roles. The role a particular instance is either to be a Scheduler component or a passive Web-Crawler Agent. It is not ideally that there exits several Schedulers in a Web-Crawler Network, since this may cause conflicts and duplicate assignments. An initial decision needs to be made if a Web-Crawler Agent is a Scheduler or not, and therefore a Boolean input parameter $initialScheduler$ is defined. The parameter $initialScheduler$ is default false, but is set to true on one instance of the KALMAN-BLAWLB.

To be able to do decisions based on updates towards the BLA-Kalman algorithm, several input parameters are defined. These are the initial mean (InitMu), initial variance (InitSigma), the observation variance (ObservationNoise) and the transition variance (TransitionNoise). These are used to initialize each Agent-Task object in Φ. These are set by the operator of the Web-Crawler Network, and the exact values would therefore be discussed in the experiments and results chapter.

The initialization phase (step 1-12)

The method presented in algorithm 2 begins on with an initialization of the Web-Crawler Agent. The initialization is started by initializing the Web-Crawler Agent register, i.e. , the set A, and a control statement that checks if the current Web-Crawler Agent instance is initialized to be the Scheduler. If this is the case, the Web-Crawler Agent register itself and waits (step 2) for all the other Web-Crawler Agents to enter step 9 and send a register request. The method that performs this blocking request at the Scheduler instance is the $waitForAgents$ method. This method waits until all the Web-Crawler Agents are registered in the Web-Crawler Network, and return pointers to each of the Web-Crawler Agents. These pointers are added to the Web-Crawler Agent register A. The execution of this method is performed by external middle-ware, that take care of the communication and handling eventual failures that may occur in the registration process. The further

details of how this middle-ware operates is seen as beyond the scope of this thesis and therefore not further elaborated.

When all agents have issued a register request, the instance iterate through all the Agent-Tasks and apply the decision method, i.e. , the method $decideNewAgentForAgentTask$, described in algorithm 4 on each. This returns a new Web-Crawler Agent pointer based on the BLA-Kalman algorithm presented in algorithm 1. The main details of this method is discussed later, but it is important to acknowledge that this decision is made on based on the initial values set on each Agent-Task object in Φ. Since all Agent-Task objects are initialized with the same values, and no feedback is returned to any of the objects, the BLA-Kalman algorithm, i.e , presented in algorithm 1, should be able to select the Web-Crawler Agents uniformly.[1]

After selecting the appropiate Web-Crawler Agent to send the Agent-Task, the method $addAgentTaskToQueueOrSendToNew$ is called. This method either adds the Agent-Task to the Schedulers own queue, or appends it to the queue on a remote Web-Crawler Agent. These operations is, similarly to the $waitForAgents$ method, controlled by the middle-ware and therefore not further elaborated.

The final step in the initialization process is step 7, where an broadcast is sent. This broadcast indicate that the initialization is finished. The broadcast contain pointers to all Web-Crawler Agents in the Web-Crawler Network and is returned by the method $registerAgentInNetworkAndWait$. The returned pointers are appended to the agent register A on each KALMAN-BLAWLB instance.

The operational phase (step 13-28)

When the initialization phase is finished, the Web-Crawler Agent instances are entering a phase where they perform the same operations. This phase ignores if the Web-Crawler Agent is initialized as a Scheduler or a 'normal' Web-Crawler Agent. The operational phase take care of the execution of the Web-Crawler specific operations, i.e. , visiting web-pages etc., in addition to add tasks to the queue if it is empty. Since the Web-Crawler Network does not have any limit on how many web-pages it should visit, the operational phase is intended, if otherwise not signalled, to run forever.

The first steps in the operational while loop, i.e. , the steps 15-18, is intended to ensure that whenever the queue is empty, the Web-Crawler Agent is in a blocking receive mode. This is created to avoid endless and CPU intensive loops whenever an empty-queue situation occur. An empty-queue situation may occur if for instance the BLA-Kalman decide to not explore the particular Web-Crawler Agent instance. This is maybe an undesired situation, but may occur. Therefore the method $waitAndRetrieveAgentTasks$ blocks the Web-Crawler Agent until it receives one or multiple Agent-Tasks.

If the queue contain Agent-Tasks, they are iterated. The iteration process polls, i.e. ,

[1] We assume that the random number function utilized in the implementation, is returning an uniform random value

retrieve and remove, an Agent-Task from the head of the FIFO queue. It then executes the $retriveAndUpdateNewAllocation$ method. This method calls an independent defined allocation method and updates the l variable in the Agent-Task object ϕ. This variable is, as explained in section 6.3.1, a identification value. The identification value indicate which particular web-page that should be downloaded on the Agent-Task referred host. An example is that if the allocation method is a scheme to decide retrieval frequencies, i.e. , the re-crawling scheme, of temporal content as explained in section 3.2. The scheme decides which web-page that is the most likely to be updated, on the Agent-Task referred host.[2] This page is assigned to the variable l in the the Agent-Task object, and is the web-page that is visited in step 21.

How the communication between the Web-Crawler Agent and the web-environment occurs, is dependent on if it is simulated or the Web-Crawler Network exists in a real web-environment. The simulated environment may only consist of a simple data-structure with with entries for each Agent-Task. These entries may indicate the response time for that particular Agent-Task or some other value related to the Agent-Task. However a real web-environment returns a web-page that need to post-processed. This post-processing can be to for instance retrieve relevant content and store it in a database. Whit this diversity of options in mind, we therefore create a method that handle the whole process, regardless of environment. This method is called $visitWebPage$, and is a blocking request method.

As explained in 5.1, the $B(\phi)$ measure the process time from the Agent-Task enters the queue, to the web-page is retrieved and appropriate post-processing is conducted. The goal is then to equalize the sum of the assignments at a given point in time. This needs to be measured for each Agent-Task and therefore a the total processing time is registered at step 22, after the blocking method $visitPage$ is finished. This value is further used as input in the BLA-Kalman update method in the decision phase.

The decision phase (step 23-24)

The decision phase, is the phase where the KALMAN-BLAWLB decide where to send the Agent-Task. But also if it want to continue to process the Agent-Task, by adding it in its own queue. The problem is shortly to find the Web-Crawler Agent most suitable for getting the best total measured process time. A such decision may be described as an algorithm often referred to as a simple problem-solving agent as described by Russel et. al [51] (page 61). In a simple problem-solving agent the decision and update is performed within a single method. However since we want to use the decision method without updating the BLA-Kalman variables in the initialization phase, we define two separate methods: $decideNewAgentForAgentTask$ and $updateBLAWithResponse$. These are displayed in algorithm 4 and 3.

The method $updateBLAWithResponse$ contains the update instructions in a very similar fashion to the standard BLA-Kalman algorithm described in algorithm 1 and section 4.2. However the main difference is instead of fixed state variables, they are con-

[2]We assume that the host is visited at-least once before, using an DAG traversal allocation scheme.

Algorithm 3 The KALMAN-BLAWLB state update algorithm

The BLA-Kalman update method

Input: ϕ_j = $(\{\mu_0[N] \dots \mu_n[N]\}, \{\sigma_0[N] \dots \sigma_n[N]\}, \sigma^2_{ob}, \sigma^2_{tr}, T_q)$; g that is the feedback measured in seconds; a_j Web-Crawler Agent suggested to perform the response

Initialization:

Method: updateBLAWithResponse()

1: $\mu_{a_j}[N+1]$ = $\frac{(\sigma^2_{a_j}[N]+\sigma^2_{tr})*g+\sigma^2_{ob}*\mu_{a_j}[N]}{\sigma^2_{a_j}[N]+\sigma^2_{tr}+\sigma^2_{ob}}$
2: $\sigma_{a_j}[N+1]^2$ = $\frac{(\sigma^2_{a_j}[N]+\sigma^2_{tr})\sigma^2_{ob}}{\sigma^2_{a_j}[N]+\sigma^2_{tr}+\sigma^2_{ob}}$
3: **for all** $a \in A$ **do**
4: **if** $a! = a_j$ **then**
5: $\sigma^2_a[N+1] = \sigma^2_a[N]$
6: $\sigma^2_a[N+1] = \sigma^2_a[N] + \sigma^2_{tr}$
7: **end if**
8: **end for**

Algorithm 4 Web-Crawler Agent selection for KALMAN-BLAWLB

The BLA-Kalman select next Web-Crawler Agent

Input: ϕ_j = $(\{\mu_0[N] \dots \mu_n[N]\}, \{\sigma_0[N] \dots \sigma_n[N]\}, \sigma^2_{ob}, \sigma^2_{tr}, T_q)$; $A = a_0 \dots a_n$ registered Web-Crawler Agents

Initialization:

Method: decideNewAgentForAgentTask()

1: **for all** $a \in A$ **do**
2: Draw a value random x_a from the associated *normal* distribution $f(x_a, \mu_a[N], \sigma_a[N])$
3: **end for**
4: Pull the Arm i whose drawn value x_a that have the smalles values of the randomly drawn values: a_r = $argmin_{a \in \{1 \dots n\}} x_a$
5: **return** a_r

tained in the Agent-Task object ϕ. This object is passed by reference to the method and the method then updates $\mu_{a_j}[N]$ and $\sigma_{a_j}[N]$. The results are kept in the object until the method ends its execution. The consequences of such usage is that the variables are out of range to the method until the same object is passed to it again. This means that each Web-Crawler Agent is in reality more or less state-less, since all of the information is passed around in Agent-Tasks.

As described in section 5.1 and further stated in section 5.2.1, one of the goals in the system is to equalize the sum of the process time for each task. The particular plan to achieve this equalization, is to minimize the process time together with trying several combinations of Agent-Tasks in the queues. This means that the BLA-Kalman decision algorithm needs to be altered to support the feedback model. Instead of se-

lecting the Web-Crawler Agent that give the largest process time , it is desirable to select the one with the lowest response. This is the 'less-is-better' reward scheme mentioned in section 5.2.1. The 'less-is-better' reward strategy is performed by using the $argmin$ method instead of the $argmax$ method used in algorithm 1. The resulting method $decideNewAgentForAgentTask$ can be viewed in algorithm 4.

6.4 Implementation

The implementation of the proof of concept solution is entirely developed in Java. The main reason for choosing Java, is the object orientation, good support for multi-threading and the native support for concurrent data-structures. This makes it relatively easy to create threads and communicate between the components. The concurrent data structures make sure that writing and reading of shared memory is properly handled and it is easy to focus on the core components of the algorithm.

To be able to separate the functionality of the components, we required the implementation to be object-oriented. The most central objects are the Web-Crawler Agent object and web-environment objects. These have , however , several additional subordinated objects associated. These objects are implemented by using Java interfaces and abstract classes. We have decided to use interfaces and a good object oriented structure to make the implementation as flexible as possible. The result was that we were able to 'plug-in' the IPMicra and Concurrent Hash algorithm with little effort.

There are several external libraries utilized in the implementation. The most crucial to mention is the Apache Commons Math library. This library provide scientific and mathematical functionality, that the native Java Application Programming Interface (API) do not provide . One of the Apache Commons Math features, i.e. important in our implementation, is the random functions that is used to sample normal distributed values. This functionality is used both by BLA-Kalman selection algorithm 4 and the RTT functions in the simulated web-environment. In addition Apache Commons Math library, is powering the sampling functions with a Mersenne Random Generating algorithm. This function is a Pseudo Random Number Generator (PRNG), suitable for simulations and Monte-Carlo analysis [2].

Since the implementation is seen as a prototype, that needs to be iteratively tested. The implementation must run in an effective manner. This means that we remove some of the communication costs that we see as relatively negligible according to our research goals. The communication cost between the Web-Crawler Agent components shown in figure 6.1, is a such negligible communication cost. Even if the Web-Crawler Agents are exchanging information between each other, this is a small overhead when considering the large amount of information that is gathered and sent to other components in the Web-Crawler Network. Therefore, to conduct an effective simulation, we build a single process, that can live entirely on a computational unit and applying multi-threading to simulate Web-Crawler Network components within this process.

Chapter 7

Experiments and Results

Chapter 7 starts with explaining the background for the simulated web-environment thoroughly. The RTT function definitions and how the geographically/topological distance is emulated in the simulation. Further the experiment settings is elaborated for each of the solution algorithms. Finally the results are presented in section 7.3.

7.1 Creating a simulated web-environment

To be able to test load-balancing algorithms, there are two essential methods mentioned in the literature [44] [38] [19]. One is to test the Web-Crawler Network in a simulated web-environment or to test it in a real web-environment. A real web-environment could be set up of a specified amount of web-hosts that need to be visited. The real environment would be give the most authentic results, but requires a lot of infrastructure and production ready code to operate. In addition the control of the environment and simulation is dependent on QOS delivered from many participants in the environment and the Web-Crawler Network. This is especially difficult when you want to test geographical distributed Web-Crawler Networks and have no particular budget to rent a world-wide network of computers. It is therefore more convenient to test KALMAN-BLAWLB in a laboratory with a simulated web-environment.

7.1.1 Pre-experiment research

A Web-Crawler have in all cases we have studied used the HTTP protocol as the primary way to communicate between the target host and the Web-Crawler Agents [10] [49] [8]. There may however be a reason to assume that other protocols like File Transfer Protocol (FTP) etc. are used, but we believe that this is not the preferred protocol in modern Web-Crawler Networks. We therefore need to build the web-environment based on responses received from HTTP requests.

To be able to create a limited web-environment it is necessary to study a limited set of hosts and RTT values from multiple requests sent towards the hosts. In this research we wanted to study standard deviation and mean values on the sets of gathered RTT values. We registered approximately 1000 requests on more than 100 various blogs and news hosts. The registrations was performed by a large scale Web-Crawler implemented in Java. Te registrations measured the the RTT from a HTTP request was sent by the Web-Crawler, to the page was downloaded and available for post processing. The blogs with the associated registered values can be found in the appendix section D.

When studying the RTT values we saw immediately that there was large differences in the mean values. It ranged from approximately 0.2 s per request to 3 to 4 s. The big difference may be caused of capacity, popularity and distance. The hosts we studied also had a relatively large standard deviation difference. Raging from 0.050 s to 1 s. A common pattern was however that the hosts that had relatively large mean, also had relatively large standard deviating on the measured values.

7.1.2 Measuring distance

As described in section 5.1.1, the distance between a Web-Crawler Agent and the host that the Agent-Task represent, is crucial in a geographical distributed Web-Crawler Network. This distance is calculated by the function $\delta(a_j, \phi)_{\phi_j}$ and should be specified by the simulation environment. We intend to make the distance function as simple as possible and recognize that the function and scheme may cause some loss of generality. However we believe that if we partition the environment into several partitions we could imagine that each partition is a country, organization or continent. When for instance HTTP request travel from partition/country 1 to partition/country 3, it covers a distance of 2 partitions. This approach is extremely simple and a high-level approach of the web-partitioning research conducted by Jos'e et. al [19]. We therefore calculate the distance by partitioning the web-environment and use each partition index as the position identifier. When we need the distance we just subtract the position of the Web-Crawler Agent and the position of the host, i.e., represented by an Agent-Task. The fully defined $\delta(a_j, \phi)_{\phi_j}$ function is presented in equation 7.1.

$$
\begin{aligned}
p_\phi &= \text{The partition where the host represented by the Agent-Task is located} \\
p_{a_j} &= \text{The partition where the Web-Crawler Agent is located}
\end{aligned}
$$

$$
\delta(a_j, \phi)_{\phi_j} = |p_\phi - p_{a_j}| \tag{7.1}
$$

7.1.3 Response functions

As mentioned in section 5.1.1 the stochastic RTT function ($g(\phi, a_j, \delta)$) is defined by the simulation environment. The property we are looking for in a such function, is that it re-

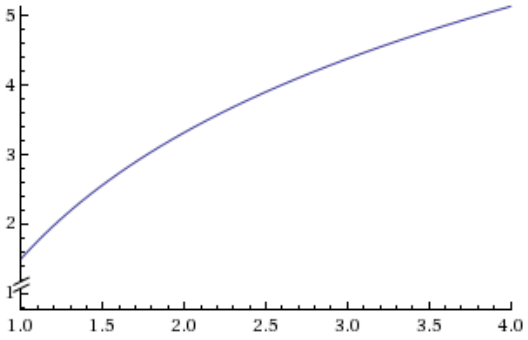


Figure 7.1: A plot showing the RTT function 7.2 ($k = 7, p = 4, \mu[\phi] = 1.5$)

turns a RTT value based on the measured values from our test set. In addition it must also include functionality that adds latency differences present in a geographical distributed Web-Crawler Network. We have therefore created a function, that returns the mean RTT value and adds latency by using the natural logarithm. The function is displayed in equation 7.2.

$$\begin{array}{lcl} k & = & \text{A confusion constant} \\ p & = & \text{The amount of partitions} \\ \mu[\phi] & = & \text{The prior measured mean value for } \phi \end{array}$$

$$g(\phi_i, a_j, \delta_{\phi a_j})) = \begin{cases} \delta_{\phi a_j} > 0, & \mu[\phi] + \frac{k}{p} * \mu[\phi] * \log(\delta_{\phi a_j}) \\ \delta_{\phi a_j} = 0, & \mu[\phi] \end{cases} \tag{7.2}$$

The function displayed in equation 7.2, is receiving the current processing Agent-Task, the current Web-Crawler Agent and the distance returned from function defined in equation 7.1 as input. The function is constructed to return the prior measured mean value, i.e., registered for the particular host/Agent-Task, when the distance is 0. This means that the Web-Crawler Agent and the host that the Agent-Task is representing is located in the same partition. Otherwise a penalty is added by multiplying a fraction of the measured $\mu[\phi]$ with the natural logarithm of the distance. We have added this logarithmic penalty to avoid a linear increasing value as the distance increases. The function should be carefully plotted before starting the simulation, to avoid unrealistic small or large differences between the partitions. An example plot is therefore shown in figure 7.1.2, where $k = 7$, $p = 4$ and $\mu[\phi] = 1.5$. The plot is between $\delta_{\phi a_j} >= 1$ and $\delta_{\phi a_j} <= 4$

As explained in section 5.1, the stochastic function X_ϕ in addition to receive $g(\phi, a_j, \delta)$ as input, it adds noise related to each Web-Crawler Agent by using the function $\sigma(\phi, a_j)$. This noise can be for instance caused by processing problems, network errors or other issues close or at the Web-Crawler Agent . However since we not to intend to simulate any capacity problems or similar, the function $\sigma(\phi, a_j)$ only returns the σ related to the hosts and response table in the appendix D.

Table 7.1: Envrionments

EnvrionmentName	Agent-Tasks	Repeated
Small	100	100
Large	1000	100

7.2 Experiments

To be able to test the solution properly, three various KALMAN-BLAWLB scenarios are considered. One scenario where little exploration is utilized , a scenario using a medium amount of exploration and one where we use massive exploration to load-balance the system. Then these settings are compared with Consistent Hashing algorithm and the IPMicra algorithm. All scenarios are tested with both 100 and 1000 Agent-Tasks, and repeated 100 times. Constantly the standard deviation of the processing time is measured in an ensemble average between the Web-Crawler Agents, together with how many Agent-Tasks it is able to process every second. In addition the total amount of exchanged Agent-Tasks are recorded for all the experiments. By performing all these experiments we should be able to tell something about how load-balanced the system is, the system utilization and if the solution is able to handle a larger amount of Agent-Tasks than the initial 100. An overview of the Agent-Tasks configurations is found in table 7.1

7.2.1 General settings

The amount of Web-Crawler Agents in the Web-Crawler Network is in the test-setup vital for any further settings. It is tempting to create a large amount of Web-Crawler Agents, and test them in a geographically distributed Web-Crawler Network. However since we only simulate the distribution and not actually intend to distribute it physically, we need to limit the amount of Web-Crawler Agents. The limitation is therefore the hardware available to our experiments. We have only three machines available to conduct our experiments on. These machines that have either two or four core CPUs. These are able to run four Web-Crawler Agent threads simultaneously with relatively little effort. It is very important that there are as few context switches during the test-process as possible. If we increase the number of threads, they could maybe cause starvation of one or more of the Web-Crawler Agent threads. This is critical, when we are dependent on cooperation of all the Web-Crawler Agents to load-balance the system. The general settings for all the test scenarios is therefore set to four Web-Crawler Agents in Web-Crawler Network.

It is mentioned several places in chapter 6, that we intend to separate the algorithms related to allocation and the KALMAN-BLAWLB. This means that we need to define an allocation algorithm that can be applied in our simulations. The choice of allocation algorithm can influence the results. Improvements to KALMAN-BLAWLB, where various allocation schemes can be integrated can be further discussed, but for the sake of brevity,

we use a simple one page allocation. This means that we only visit one selected web-page below each host, when the Agent-Task is processed.

The time each test need to be conducted, is obviously something that is critical, since we need to at least test each configuration 100 times to get reasonable and conclusive results. It is tempting to run the tests iteratively, with no limits or pauses when processing the Agent-Tasks. But, as explained in section 5.1.4 and in section 5.2.1, the problem need to be dynamically solved by time and by cooperation. This means that each Web-Crawler Agent have a time-line and need work in real-time. For instance actions performed by Web-Crawler Agent 1 at t may affect Web-Crawler Agent 2 at time $t + 2$. If Web-Crawler Agent 2 just processes the Agent-Task without taking a pause, i.e., stochastic determined, the Web-Crawler Agent 1 and Web-Crawler Agent 2 may be aligned differently than in a real life situation. Therefore the Web-Crawler Agents need to be tested in real-time, but we allow us self to increase the clock speed 50 times. In our reality 1 second is therefore 20 ms.

7.2.2 Environment settings

How we want to configure the environment is important for how the results are interpreted and how 'easy' it is to solve for the algorithms. A relatively logical configuration to grasp, when we dealing with geographical distributed Web-Crawler Networks is to separate the environment into continents. We can pretend that partition 1 is Africa , partition 2 is Asia and Australia, partition 3 is Europa and partition 4 is America. Let say that the crawler is focusing their effort on European and American hosts, the majority of the hosts would be closest in distance from the European and American located Web-Crawler Agents. This is realistic situation in for example focused crawling towards European and American social media hosts. We therefore use the following distribution in the environment containing 100 hosts: Partition 1 contains 6 hosts, partition 2 contains 13, partition 3 contains 25 and partition 4 contains 56 hosts. In the environments containing 1000 hosts, the values is multiplied by 10. All the settings are displayed in table 7.2.

It is a danger, when you have relatively many fast hosts and many slow hosts, that a single partition could contain too many of one type. However it is not that unrealistic either. A segment of the web, could for instance be placed in a low latency network and running on high capacity servers. However to make the environment fair for all the contenders, we regenerate the environment by random shuffle the hosts 100 times. Each test run, is then tested against the same environment. Test-run 1 for the consistent hash load-balancing algorithm is for instance tested against the same environment, i.e. , environment permutation 1, as the IPMicra test-run 1.

The environment selected to test the KALMAN-BLAWLB is a relative small subset of the large web. In addition we need to pick the hosts that we wanted to represent as Agent-Tasks carefully. We need a mix of both slow hosts and fast hosts. However, since we increased the tempo of the simulation artificially we need to sacrifice the fastest hosts. The simulation is instead of downloading a web-page, to pause the Web-Crawler Agent

Table 7.2: Hosts placed in the partitions

Distribution	EnvironmentName	P1	P2	P3	P4
Non-uniform	Small	6	13	25	56
Non-uniformLarge	Large	60	130	250	560

Table 7.3: The simulation environment settings, utilized when testing the scenarios

Envrionment-Description	k (see eq 7.2)	p (see eq 7.2)
Non-uniform	7	4
Non-uniformLarge	7	4

for a specified amount of seconds. If this value is below 1 ms, the Web-Crawler Agent does not wait and just continue to operate. If this happens repeatedly the Web-Crawler Agent may increase its processing rate falsely. When increasing the speed 50 times, this means that all response values below 50 ms is increasing the process rate falsely. Therefore the hosts that have a high possibility of returning a 50 ms RTT or less removed from the environment set.

7.2.3 KALMAN-BLAWLB

As described earlier we intend to test three scenarios of KALMAN-BLAWLB settings in the Web-Crawler Network. The three scenarios are related to how much exploration we use in our experiments. The three scenarios are little exploration, medium exploration and one where we use massive exploration. However, to determine the exact parameters we set on each Agent-Task, i.e. , the parameters $ObservationNoise$ (σ_{ob}) and $TransitionNoise$ (σ_{ob}) in algorithm 2, is difficult. We do not exactly know the reward differences, as in the experiments conducted by Granmo and Berg, prior to the experiments [23]. It is also very inconvenient to determine $ObservationNoise$ (σ_{ob}) and $TransitionNoise$ (σ_{tr}) for all the, i.e. , 100 or 1000, Agent-Tasks individually. In addition we need to determine the parameters described in algorithm 2, $initMu$ and $initSigma$. We tried to conduct a singe test-run, where the partitions was not heavy skewed, i.e., the hosts is uniformly distributed across the partitions. This showed that the reward differences varied from each Agent-Task, so a common value was very difficult to determine. However as shown in the research performed by Granmo and Berg, the BLA-Kalman performs reasonable well also by missing the 'correct' $TransitionNoise$ (σ_{ob}) and $ObservationNoise$ (σ_{ob}) $TransitionNoise$ (σ_{tr}) values. We therefore experimented with various values described in table 7.4, but can not exactly pinpoint how much they diverge from the 'true' reward difference.

Table 7.4: The various scenario KALMAN-BLAWLB settings

Experiment	σ_{tr}	σ_{ob}	EnvironmentName	$initMu$	$initSigma$	Exploration
1	0.1	2.0	Small	0.1	1000	Little
2	2	4	Small	0.1	1000	Massive
3	0.1	20	Small	0.1	1000	Moderate
4	0.1	2.0	Large	0.1	1000	Little
5	2	4	Large	0.1	1000	Massive
6	1.0	20	Large	0.1	1000	Moderate

There are several plots found in the appendix section E showing how the $\sigma[N]^2$ converges with the various parameters. The plots show that all experiments approaches a constant $\sigma[N]^2$ relatively fast, and that they are relatively low. The plots for the parameters utilized in experiment 3 and 6, show that $\sigma[N]^2$ is relatively slower to approach a value than the other settings. This would therefore give a larger opportunity to experiment during the initial 100 iterations each Agent-Task is processed in experiment 3 and 6.

The classification of the Little, Massive and Moderate exploration is based on research we have conducted on the parameters found in table 7.4. For instance, we tested how $\mu[N]$ behaved on the various settings when pulling values from a stationary and normal distributed environment. The plots are found in the appendix section G. These plots shows that experiment settings 1,3,4,6 have relatively little exploration, compared with 2 and 5. We can observe this, because of the relatively dispersed plot in figure G. In addition the σ_{tr} is used in the BLA-Kalman algorithm to increase the $\sigma[N]$ for all Web-Crawler Agents, i.e. , step 6 in algorithm 3. This means that the slight larger set σ_{tr} in experiment 2 and 5 is encouraging exploring of the Web-Crawler Agents that is not visited. However the experiment settings in 3 and 6 would probably experiment more because the large $\sigma[N]$ in the initial iterations, seen in the plot in figure E.2. We therefore classify the exploration settings, used in experiments 2 and 5, to be our Massive scenario. The settings used in 3 and 6 are accordingly classified as Moderate.

The $initMu$ and $initSigma$ parameters need also to be set to some reasonable initial values. Unpublished research conducted by Berg, i.e. , one of the authors of the initial BLA-Kalman, have revealed that the size of $initSigma$ parameter have little or no significance of how the BLA-Kalman converges towards a stationary mean and variance. A few plots on how this convergence occurs are found in the appendix section D. The plots show that the $\sigma[N]$ converges all-ready before 5 iterations of sampling is conducted, with $initSigma$ set to 60 and 1000. The $initSigma$ value should however be set sufficiently large to obtain number a more accurate first guess on $\mu[N]$. We therefore have suggested to set the $initSigma$ value to 1000 as a standard value and $initMu$ to a rather low value 0.1.

7.2.4 IPMicra

The IPMicra solution is as explained in section 3.3.2, a solution that utilize IP-subnets to decide the distribution of the Agent-Tasks. The environment we have described and defined is not a IP-based environment. The IPMicra solution does however require each host to have a real-world IP address, but they need to belong to a subnet hierarchy. We have therefore created a subnet hierarchy of 27 subnets, with a total of 20 leaf subsets.

The existing environment is logical attached to the leaf subnets. Each subnet, ,for the sake of brevity, contain 5 or 50 hosts. The hosts in each subnet are however ensured to belong to either the same partition or the neighbouring partition. This ensures that the RTT retrieved from the function described in section 7.1.3 is relatively similar to all the hosts found in the same subnet.

It is also important to mention that IPMicra does not use the $B(\phi, a_j)$ like KALMAN-BLAWLB. This means that it does not consider the total process time of the Agent-Tasks, but only the RTT received from the function $g(\phi_i, a_j, \delta_{\phi_{a_j}}))$. This must later be considered when discussing the results from the experiments

The only described parameter in the IPMicra algorithm, is the response threshold. The response threshold is utilized when deciding from where it is fastest to process a Agent-Task. We set this threshold to 2.0 s. But it is also described, outside the algorithm description, that there are performed continuously statistical measurements to identify if the subnet is attached to wrong Web-Crawler Agent. This part of the algorithm need a threshold value to determine how many Agent-Tasks that is below the process time threshold. This value is set to 5 when testing in a **Small** environment or 25 when testing in the **Large**. This means that when 5 or 25 Agent-Tasks are visited and the process time is above the threshold, the subnet need to re-assigned to another Web-Crawler Agent.

7.2.5 Consistent Hash-scheme

The consistent hashing scheme do only have one single parameter to adjust. This is as explained in section 3.3.1, the amount of buckets present in the consistent hashing circle. The parameter is referred to as the amount of replicas in the consistent hashing circle. We see this as a very trivial parameter, and we only need to ensure that it is sufficiently large compared with the amount of Agent-Tasks. We therefore set the amount of replicas to 100.

7.3 Results

To be able to present results we measure constantly by registering information in a large database. The information is building a large historical log of how the prototype Web-Crawler Network operate. Parameters like time, communication, throughput and the

RTTs etc. are constantly measured. This data is heavily post-processed and structured to present an ensemble average time-line of how the Web-Crawler Network perform. The time-line is measured in seconds. In addition the time-line is limited to 150 000 seconds or almost 42 hours. We limited the time-line, to make the time-line computation and the experiments less time consuming. In addition we noticed, by performing inital tests, that the measured values converged enough to support our research questions within this time limit. To be able to measure the effect and scalability of our solution we therefore made three distinct measurements of the time-line: Fair load-balancing, system utilization and communication.

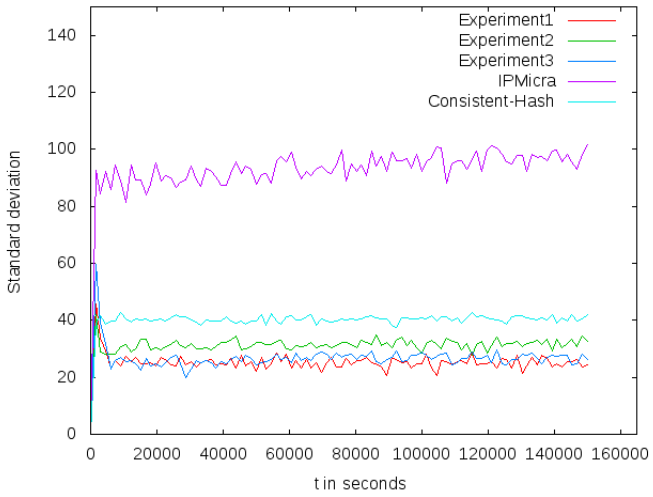
The plotted results, even if repeated over 100 times, turned up to be relatively noisy. This caused problems when we wanted to compare the experiments that differed by a very small margin. However the results from the competing algorithms compared with the KALMAN-BLAWLB show clear differences despite the noise. We therefore present the plots as a cubic-spline interpolated curve. The cubic-spline interpolation keep most of the shape of the curve, but remove the worst noise spikes. Each experiment is in addition plotted without using any kind of interpolation curve. These can be found in the Appendix section F. The fairness measurements are presented together with a correct comparison plot. The comparison plot contain the best KALMAN-BLAWLB experiment and the IPMicra and Consistent-Hash plots.

7.3.1 Fair load-balancing

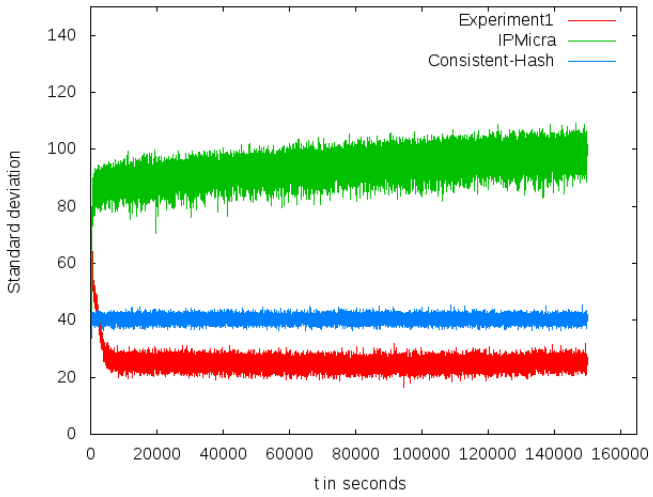
To measure load-balancing, it is convenient to use some kind of fairness scale to see if the process time is reasonable equalized. Multiple methods to measure fairness in load-balancing schemes are described in the literature . The most accurate and easy to grasp is the fairness index. However since the environments we have created do not have any optimal solution approach , it is very difficult to find the optimal process-time at time t. This makes the fairness index impossible to calculate, since it relies on having the optimal process-time pre-calculated. We therefore selected the well known standard deviation, i.e. , often referred to as σ, as the fairness measurement method. How the standard deviation is calculated at time t is displayed in equation 7.3.[30]

$$\sigma_t = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(next(B(\phi, a_j), Q_i, t) - \overline{B(t)})^2} \tag{7.3}$$

The standard deviation is not a normalized index showing a value between 0 and 1 as the traditional fairness index. It is instead showing a value of how much the Web-Crawler Agents deviate from each other in process time. What this means is that if Web-Crawler Agent1 is processing an Agent-Task at time t The most desired value is obviously 0. This indicate that the load is totally equalized on all Web-Crawler Agents. We are therefore looking for the a trend in the plotted results that approaches 0 or is non-increasing.



(a) Cubic-spline interpolated plot



(b) Correct comparison plot

Figure 7.2: The figure (a) and (b) shows the standard deviation of the processing time between all the Web-Crawler Agents in the Small environment.

Small environment

In the plot presented in figure 7.2(a) and 7.2(b), we see several interesting details. The first is that all algorithms seem to stabilize within the time limit. However the IPMicra have a slight increasing curve throughout the whole test-period, and the standard deviation measure is much higher compared with the other algorithms. The Consistent-Hash algorithm performs , according to the plot, slightly worse than the KALMAN-BLAWLB algorithm. The Consistent-Hash is however the most stable algorithm, comparing with all other tested algorithms. The Consistent-Hash algorithm have no learning, and therefore stable from the first second.

If we look at the KALMAN-BLAWLB plots, the KALMAN-BLAWLB have clearly a learning phase where it not load-balance the Web-Crawler Network fairly. This learning seem to be conducted within the first seconds or 3 hours in all the KALMAN-BLAWLB experiments. Experiment1, i.e. , the experiment with lowest exploration, seem to learn slightly slower but manages to load-balance the Web-Crawler Network slightly better when the load-balancing stabilizes. This is substantially better than the IPMicra, that seem to learn to load-balance the Web-Crawler Network continuously in a wrong direction. However the static Consistent-Hash algorithm does not employ any learning to achieve a standard deviation value and therefore is able to perform a fairly load-balanced Web-Crawler Network from the first second.

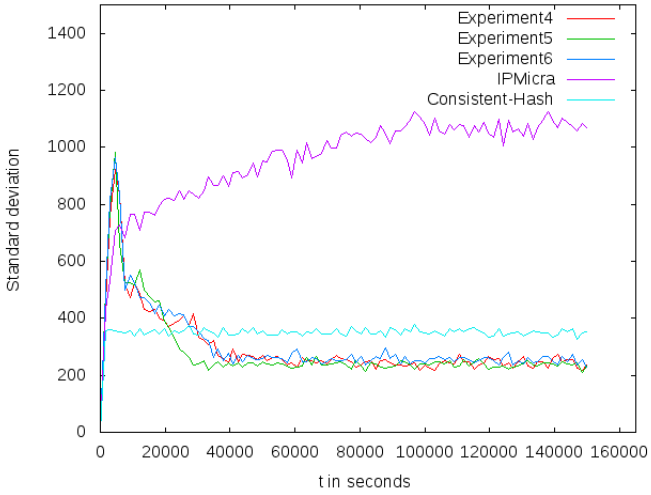
If we only focuses on the various KALMAN-BLAWLB plots in figure 7.2(a), we see that the parameter difference do not have a very large influence on how the standard deviation is plotted. A slight difference can however be made between the experiments. Experiment1 and Experiment3 are almost indistinguishable the first 25 000 seconds. Then it seem to explore the Web-Crawler Network in a slight unfavourable manner. This is however a small difference, compared with Experiment2. Experiment 2 approaches Experiment1 the first 5000 seconds, but it seem not reach the stable state of Experiment1. Instead the KALMAN-BLAWLB continuous to explore the Web-Crawler Agents and stabilize on a notable higher standard deviation.

Large environment

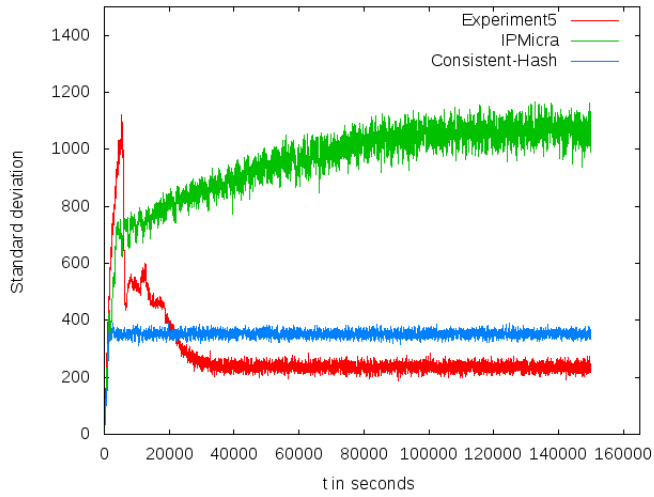
The plot presented in figure 7.3(a) shows that the standard deviation is much higher when increasing the amount of Agent-Tasks to 1000 in the Web-Crawler Network. The Consistent-Hashing and KALMAN-BLAWLB stabilize on approximately 200 - 400s. The Consistent-Hash algorithm is however the most stable during the 40 000 first seconds, then the KALMAN-BLAWLB seem to stabilize as well. It is also notable that IPMicra seem to be able, even if the plot is very noisy, to stabilize between 1000 and 1200. This stabilization was the IPMicra algorithm not able to perform in the smaller environment.

When we look at the learning separately, we see the learning phase is lasting much longer than in the experiment plots presented in figure 7.2(a). The KALMAN-BLAWLB learning seem to take almost 40000 seconds, or nearly 11 hours before it seem to converge towards and stable solution. The IPMicra is however use almost 80000 seconds to be able to learn an stabilize. The 7.3(b) reveal that the Consistent-Hashing seem to have a learning phase. This is really fast and probably related to that the process-time is continuously increasing value during the start-up before it stabilizes.

If we only focuses on the KALMAN-BLAWLB plots in figure 7.3(a), they show that the learning phase is almost identical in all experiments. The small variation, is that the experiment employing little exploration is converging faster. Experiment4 start to converge around 30 000, that is circa 10 000 seconds faster than Experiment5 and Experiment6. However they seem to be indistinguishable when stabilizing, at a standard



(a) Cubic-spline interpolated plot



(b) Correct comparison plot

Figure 7.3: The figure (a) and (b) shows the standard deviation of the processing time between all the Web-Crawler Agents in the Large environment.

deviation of 250 - 300s.

7.3.2 System utilization

It is interesting to investigate the fair distribution of the Agent-Tasks in the Web-Crawler Network. But is it increasing the overall efficiency of the Web-Crawler Network? A trivial method of measuring the efficiency of the Web-Crawler Network, is to record the overall system utilization. The definition of system utilization is in our case is relatively straight forward. The amount of Agent-Tasks processed per second. This type of system utilization have in addition been measured and compared with the fairness in the literature

earlier [44]. We therefore post process the time-line to investigate how many Agent-Tasks the Web-Crawler Network can process per second.

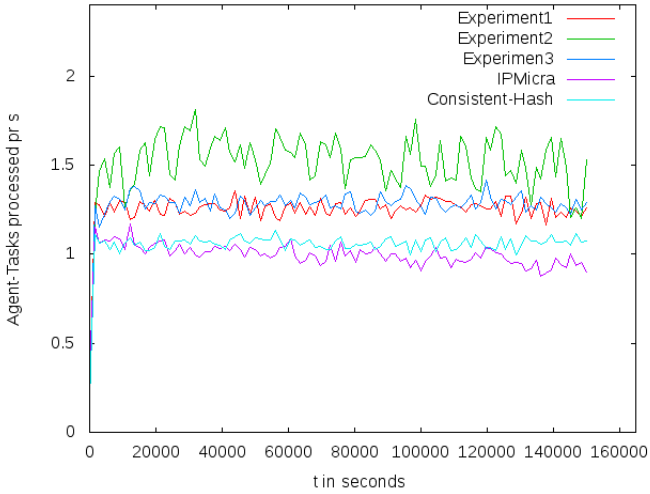


Figure 7.4: The figures shows the plot of processed Agent-Tasks per second for the whole Web-Crawler Network. The environment tested is the Small environment, that consists of 100 Agent-Tasks (Cubic spline interpolated plot)

Small environment

The plot in figure 7.4, show how the various algorithms that load-balance the Web-Crawler Network causes a relatively equal system-utilization. Both Consistent-Hashing and KALMAN-BLAWLB is able to stabilize the system-utilization and keep it steady throughout the test-period for two of the settings. It is however worth noticing that IP-Micra seem to have a decreasing curve, and is not able to keep the system utilization steady. Experiment2 is also performing better than all other tests, but the measurements are extremely unsteady and noisy.

The difference between the KALMAN-BLAWLB and the other algorithms are overall not that large. The IPMicra and Consistent-Hashing is able to keep the system relatively firmly around 1 - 1.1 Agent-Task per second. The KALMAN-BLAWLB is , however , able to load-balance the Agent-Tasks in a clever manner and stabilizes on above 1.2 - 1.3 Agent-Tasks per seconds and the noisy Experiment2 around 1.5. In the perspective of 24 hours, the difference between retrieving 1 Agent-Task per second and 1.2 Agent-Task per second is 17280 Agent-Tasks.

When studying the KALMAN-BLAWLB separately, the learning phase is not clearly recognizable in the plot. There is however, if closely examined, a steep learning curve approximately the first hour and a slight increasing trend before it stabilizes. These seem almost identical on all the parameter variants of the KALMAN-BLAWLB.

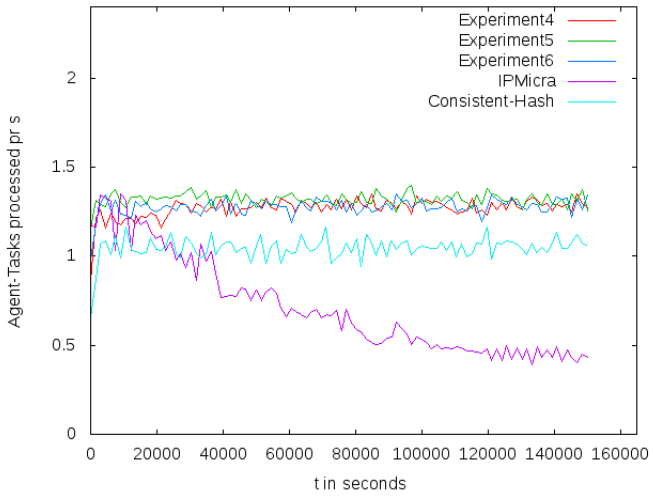


Figure 7.5: The figures shows the plot of processed Agent-Tasks per second for the whole Web-Crawler Network. The environment tested is the Large environment, that consists of 1000 Agent-Tasks (Cubic spline interpolated plot)

Large environment

The plot in figure 7.5 show the system utilization when increasing the amount of Agent-Tasks in the Web-Crawler Network. The plot is relatively identical to the plot shown in figure 7.4. The KALMAN-BLAWLB seem to stabilize the system-utilization at around 1.2 - 1.3 Agent-Tasks per second. The Consistent-Hash algorithm seem to converge towards the same value as presented in the plot in figure 7.4. I could maybe be argued that it is slight lower, but this can not be stated exactly. However the IPMicra solution seem to not be able to stabilize the system utilization in any sense. The plot is showing that after being able to have a relatively good start, the IPMicra is gradually decreasing the system-utilization.

The KALMAN-BLAWLB solution seem to stabilize on 1.2 - 1.3 Agent-Tasks per seconds and the Consistent-Hash 1 to 1.1. This is almost the same as the measurements conducted on the tests with 100 Agent-Tasks. The IPMicra is however outperformed in this test-case, by both Consistent-Hash and KALMAN-BLAWLB.

If we only focuses on the KALMAN-BLAWLB plots, we observe that they are as close as in the previous test with 100 Agent-Tasks.

7.3.3 Internal communication

The last measurement we investigate is the communication between the Web-Crawler Agents in the Web-Crawler Network. As earlier mentioned we do not particularly see the this traffic as a large problem, since there are a substantial other types of data-flow in

Table 7.5: The KALMAN-BLAWLB settings

Experiment	σ_{tr}	σ_{ob}	Environment	Exchanges
1	0.1	2.0	Small	1795.49
2	2	4	Small	46242
3	1.0	20	Small	25715.58
4	0.1	2.0	Large	14542.13
5	2	4	Large	12131.8
6	1.0	20	Large	19726
CONSISTENT-HASH	N/A	N/A	Small	0
CONSISTENT-HASH	N/A	N/A	Large	0
IPMicra	N/A	N/A	Small	92.78
IPMicra	N/A	N/A	Large	1046.43

the Web-Crawler Network. But since the previous test have shown that the KALMAN-BLAWLB perform almost equally regardless of transition and observation noise settings, it may be interesting to differentiate the experiments by looking at the amount of Agent-Task exchanges.

One Agent-Task exchange occurs when an Agent-Task is transferred from a Web-Crawler Agent to another. The measurements of these exchanges are presented in table 7.5. The measurements show that the number of exchanges increases together with the exploration when load-balancing the Small set of Agent-Tasks. However when load-balancing the Large Agent-Task set, there are very little difference between the amount of exchanged Agent-Tasks. It is also notable that Consistent-Hash and IPMicra have no or very few exchanges between the Web-Crawler Agents.

Chapter 8

Discussion and summary of results

In chapter 8 we first discuss the test-settings and how chosen environment sizes have an impact on the achieved results. Further the results are discussed according to the three main research aspects: fair load-balancing, system utilization and scalability. Lastly, we discuss some thoughts considering the whole solution approach and problems with the KALMAN-BLAWLB that need to be solved.

8.1 Test-settings

One of the most important decisions related to how the results from KALMAN-BLAWLB can be interpreted, is the complexity of the basis problem we try to solve. As stated in section 2.2, the 'NP-hardness' of the problem is determined by how many Agent-Tasks you need to load-balance in the system. In the two chosen environments, we have 100 or 1000 Agent-Tasks. It may be argued that the sets are too small to create a NP-hard problem, for the KALMAN-BLAWLB to solve. But since the value associated with each Agent-Task is stochastic and non-stationary, the problem intuitively get harder to solve . We also argue, that it is impossible to verify if a combination of Agent-Tasks is the best combination, because of the stochastic values. This is an important criterion in determinations of any NP-hard problem. This reasoning is however not backed up by solid complexity analysis, but we believe that the problem can be stated as NP-hard because of the stochastic and non-stationary nature of the problem.

The BLA-Kalman algorithm was tested against a non-stationary environment with a single agent using 2000 iterations. In our environments there exists 100 or 1000 distinct BLA-Kalman agents , i.e. , the Agent-Tasks, working in cooperation. To perform equivalent tests with KALMAN-BLAWLB, all Agent-Tasks need to receive at least 2000 updates. This conflicts with the time-limit set for the algorithm and that the Web-Crawler Agents may operate on various speeds. The time-limit causes many Agent-Tasks to not receive 2000 updates. However, this is not necessarily true for all Agent-Tasks, since the the Web-Crawler Agents may operate on various speeds. This means that Agent-Task

1 may receive more updates, because it is constantly assigned to a faster Web-Crawler Agent. This , however, is not a desired situation, but may occur. The number of iterations is not entirely sufficient, and maybe unfairly distributed to the Agent-Tasks. But as the tests show, the KALMAN-BLAWLB seem to converge against stable system utilization and fairness values. It therefore seems like the number of iterations are sufficient in the environments, the KALMAN-BLAWLB is tested against.

8.2 Results

The results presented in section 7.3 are very noisy. The noise is however caused by very small spikes from the mean point on the graph. We was not able to minimize it entirely, by performing 100 ensemble averages on each test. The noise also seem to increase for the solutions that utilize learning. This may be caused by the impact the solutions creates when a new Agent-Task is added to the Web-Crawler Agent queue. In addition, it seems to be less noise in the measurements from the Large environment. We think this is caused by a lower degree of transition noise in the Large environment, and further discuss this in section 8.2.2 However, behind the noise, are very clear trends that are discussed and analysed below.

8.2.1 Fair load-balancing

As seen in figure 7.2(a) and 7.3(a), KALMAN-BLAWLB stabilizes the fairness, i.e. , the standard deviation, relatively fast in the small environment, but need to learn a lot more to stabilize in the larger environment. This can be caused by increased problem complexity when adding more Agent-Tasks. It also reveals that the learning scheme may have a weakness. When you add the total amount of Agent-Tasks in the system, the queues will also increase in size. This means that the KALMAN-BLAWLB algorithm, assigns an Agent-Task to the Web-Crawler Agent queue and receives an update from performing this decision a long time later. When adding even more Agent-Tasks, this delay can in worst case increase linearly. This means that the learning will be slower and more difficult when adding more Agent-Tasks. In addition the complexity of the problem is increasing, since the amount of Agent-Tasks is related to the 'NP-hardness' of the load-balancing problem. However all the KALMAN-BLAWLB experiments seem to stabilize within an acceptable time-limit, but it is difficult to conclude if the learning increases in a pattern by only performing experiments with two environment sizes.

The load-balancing results described section 7.3.1 (figure 7.2(a) and 7.3(a)) show that the standard deviation of the process time is stabilized by the KALMAN-BLAWLB and Consistent-Hash solutions. The IPMicra is however not stabilized and seem to increase continuously within the limited time-period in both environments. IPMicra bases its-self only on measurements of RTT between the Web-Crawler Agent and the host. Compared with KALMAN-BLAWLB, IPMicra does not include process time. The consequences

of following a RTT response only strategy, is that in our environment the load-balancing can be focused towards single Web-Crawler Agent. By this we mean that a Web-Crawler Agent located 'closest' to the majority of the hosts are getting assigned the majority of the Agent-Tasks. We believe this greedy strategy is what causes the IPMicra algorithm to not stabilize and perform worse load-balancing than all competitors. However we are not entirely sure that the IPMicra parameters are entirely perfect for the simulated environment. The response threshold is for example set statically to 2.0 s. The 2.0s value can for instance be to low for a subnet, that have several hosts with high RTT. But it could probably be argued that if we need to have correct knowledge of a threshold value, and the algorithm is heavily dependent on this threshold, the algorithm may also perform worse in many other types of environments. The stabilization issues with IPMicra is therefore not that unrealistic, but may be caused by the statically set parameters.

The different KALMAN-BLAWLB parameters seem to cause insignificant differences on the fair load-balancing (figure 7.2(a) and 7.3(a)). The only experiment that can be distinguished from the other, is the experiment with increased σ_{tr} (experiment 2). This setting is able to load-balance less fairly than the other KALMAN-BLAWLB experiments in the Small environment. This is probably caused by the relative large amount of transition noise, combined with that the parameter σ_{tr} and σ_{ob} are closer in value. This means that the received and noisy feedback is more emphasised. This is further confirmed when scaling up to 1000 Agent-Tasks. Because of reasons we will discuss in section 8.2.2, the transition noise is lower in the Large environment and therefore also less issues will occur when selecting σ_{tr} and σ_{ob} that is too close in value. This fact is also reflected in experiment 5 (figure 7.3(a)), i.e. , the experiment with similar settings as experiment 2. Experiment 5 is able to load-balance the Web-Crawler Network similarly to the other KALMAN-BLAWLB experiments. Therefore, the parameters have very little impact on the load-balancing, but it is recommended that σ_{tr} is substantially lower than σ_{ob} to support all environments.

The general fairness, i.e. , the standard deviation, is in the small environment is measured between 20 - 80 seconds (figure 7.2(a)) and in the large environment 200 - 1200 seconds (figure 7.3(a)). The span between the values in the large and small environment, show the problem with the use of standard deviation as a fairness value. Standard deviation does not reflect the fairness equally between different test-setups. This means that standard deviation stabilizes differently, when the amount of Agent-Tasks are increased. This is based on the simple reason that the general process time increases when adding more Agent-Tasks to the queue. In addition the KALMAN-BLAWLB results show that there probably is an lower equilibrium in both environments. This equilibrium is reflecting that it is not possible, with the settings used in the experiment, to approach 100 percent fairness, i.e. , 0 standard deviation. The KALMAN-BLAWLB seem to however to be able to find the equilibrium and since it is a cooperative multi-agent problem, this is probably a Nash equilibrium. Therefore, the fairness results must be read in the context of the environment it is tested in and that it is probably not possible to achieve 100 percent fairness in the environments tested.

8.2.2 System utilization

The general system utilization level, displayed in figure 7.4 and 7.5, could be perceived as very low for a Web-Crawler Network with four Web-Crawler Agents. A stabilization of only 1.2 Agent-Tasks per second is not impressing at the first glace. However, the web-environment described in section 7.1 contain mostly Agent-Tasks that have a process time above 1 second. In addition you have the impact of selecting wrong Web-Crawler Agent in the exploration of the system. This causes the system utilization to be rather low, but realistic in the chosen environment context.

As shown in figure 7.4 and 7.5, KALMAN-BLAWLB seem to stabilize the system utilization above the competitor IPMicra and Consistent-Hashing. The system utilization seem however not to be that influenced by the fair load-balancing discussed in section 8.2.1. This shows that greater achieved fairness is not directly associated with better system utilization. System utilization can be greedily increased in a geographical distributed Web-Crawler Network by only selecting the Web-Crawler Agent that is located closest to the host associated with the Agent-Task. This would however not be fair to each Agent-Task by many reasons. One is that, even if the system utilization is better, the process frequency for each Agent-Task may be very unfair. This means that a greedy approach would create a very large concentration of the Agent-Tasks on a single Web-Crawler Agent. This causes the retrieval rate to be very high, but causes the distribution to be unfair. This explains how IPMicra is able to have very decent system utilization in the Small environment, but very unfair load-balancing.

The various settings utilized in the KALMAN-BLAWLB experiments seem not to have too much impact on the system utilization. The only real impact is caused by Experiment2. This particular setting, as indicated by figure E.2, have a larger exploration and is able to detect more influence from the geographical environment. The massive amount of exchanges of Agent-Tasks shown in the table 7.5 during Experiment2, also emphasize this assumption. In addition, this show that there are a clear trade-off between system utilization and fairness and the amount of communications between the Web-Crawler Agents. However this is not shown clear in the Large environment. We believe this is caused by the geographical difference is not registered in the same degree by KALMAN-BLAWLB. The queue time T_q, from equation 5.2, increases because of the queue is larger. The RTT component, i.e. , the value returned from $g(\phi, a_j, \delta_\phi)$, is then too small to have a real impact and the transition noise is lowered in this environment. The KALMAN-BLAWLB settings is therefore almost indistinguishable in the Large environment.

8.2.3 Scalability

When summarizing all results from section 7.3, we see that the KALMAN-BLAWLB is able to stabilize the fairness and system utilization, in both the Large and Small environment. However, we see less impact on performing more exploration in the Large environment. We believe that this caused by a less visible noise in the Large environment.

The collective load-balancing is however able to keep the system utilization relatively equal in both environments. This shows that the KALMAN-BLAWLB properly is finding a good trade-off between fair load-balancing and system utilization when adding more Agent-Tasks to the solution.

When analysing the learning phase in both the Large and Small environment, it is clear that the Large environments is much more time consuming to load-balance. We do not think this is caused by the various KALMAN-BLAWLB settings, because the learning phase and the amount of communication exchanges is almost similar. It is rather caused by the extra delay between the decision and the update that occurs when having larger queues. When this delay increases, it take longer time to learn the impact of the assignment decision. We however recognize this weakness and argues that since the KALMAN-BLAWLB is able to keep a stable system utilization, the increased learning time is not critical.

8.3 General usage of the KALMAN-BLAWLB algorithm

Since we have tested KALMAN-BLAWLB in a Web-Crawler Network the algorithm is very domain specific. However, an Agent-Task is in any general scheduling problem referred to as a task or job. If in addition the process time for the job is stochastic and normal distributed, the KALMAN-BLAWLB could be applicable. It is however important to remember that the KALMAN-BLAWLB is an all dynamic scheduling algorithm, that constantly performs load-balancing. This have a certain overhead, compared with a more static solution. Therefore the domain and problem should be carefully investigated before applying a KALMAN-BLAWLB approach.

As mentioned throughout the thesis the allocation and scheduling, i.e., the load-balancing, is defined as separated algorithm. This is done because the KALMAN-BLAWLB is intended to be general as possible. The combination with an allocation technique could be the next step in further research. We believe that the KALMAN-BLAWLB is a decent start for further extensions, since we use such basic measurements as RTT and queue time. These can essentially be measured in most allocation schemes. If for example combining the breadth-first search allocation, as described in section 3.2, and the KALMAN-BLAWLB the RTT could be measured for the whole search process. This causes the measured values to be entirely different in size, but still applicable as update values to the BLA-Kalman update scheme.

8.4 Elements not covered by KALMAN-BLAWLB

There are a couple of areas that is not tested and considered by the KALMAN-BLAWLB algorithm. One is if a Web-Crawler Agent fails, and the Agent-Tasks need to be redistributed. Since the Agent-Task object described in section 6.3.1 contain redundant state

information related to all Web-Crawler Agents, a reconfiguration based on this should not be a problem. If the BLA-Kalman algorithm selects a Web-Crawler Agent that have failed, the next best Web-Crawler Agent is found by either suspending or removing the faulty Web-Crawler Agent associated state information. However, since the solution is decentralized, Agent-Tasks state information may be lost when a Web-Crawler Agent fails. This could be solved by a creating a central or decentralized Agent-State register. There are many issues with such registers and failure detecting in distributed system, but the KALMAN-BLAWLB should be able to handle failures in the Web-Crawler Network by performing a few changes to the algorithm. Future research would therefore be to adapt the algorithm to be failure resilient and test how KALMAN-BLAWLB is able to redistribute Agent-Tasks when failures occur.

Most of the competing algorithms are tested in a real environment, using several distributed Web-Crawler Agents. It could be very interesting to also place KALMAN-BLAWLB in a real web-environment, and preferably a geographically distributed Web-Crawler Network. However to be able to test in a real environment, the earlier mentioned failure handling must be implemented and tested. Further we have only tested a scale up approach, with four Web-Crawler Agents in this thesis. Therefore a scale out test should be performed, to see how the distribution is changed by KALMAN-BLAWLB when a new Web-Crawler Agent is added to the network. This test also implicates implementation of code that handle the information propagation that is needed to inform the Web-Crawler Network of the new Web-Crawler Agent.

Chapter 9

Conclusion and further work

In this thesis we have investigated if the BLA-Kalman algorithm, can be utilized to load-balance non-stationary Web-Crawler Networks. We have conducted thorough background investigation of different subjects and problems associated with load-balancing and scheduling, Web-Crawler Networks and machine learning. Further, the domains of the different problems are coupled and a prototype algorithm is designed and implemented. The algorithm is named KALMAN-BLAWLB and is a decentralized and dynamic load-balancing algorithm. KALMAN-BLAWLB is empirically tested in a geographically distributed Web-Crawler Network and compared with two competing algorithms. The experiments are conducted in a simulated web-environment. The experiments conducted is intended to show if the KALMAN-BLAWLB is capable of load-balancing Web-Crawler Networks fairly. In addition we investigate the system utilization and how well the KALMAN-BLAWLB scale.

The results obtained from the empirical tests, show us that KALMAN-BLAWLB is able to stabilize and fairly load-balance the Web-Crawler Network. The results also show that KALMAN-BLAWLB is able to obtain a more fair load-balancing than the competitors Consistent-Hash and the IPMicra algorithms. It particularly performs better than the IPMicra heuristic, that is not able to stabilize the load-balancing within the test-interval. Compared with the static Consistent-Hash algorithm, the KALMAN-BLAWLB need to perform learning to be able to stabilize in an equilibrium. However, this learning is limited to a small time interval, in a larger perspective. KALMAN-BLAWLB also outperforms the Consistent-Hash algorithm in the fairness measurements. The various KALMAN-BLAWLB input-parameters, seem to have small impact on the load-balancing. But the input parameter σ_{tr} should be set substantially lower than σ_{ob} to support both smaller and larger environments. The KALMAN-BLAWLB is therefore able to fairly load-balance a geographically distributed Web-Crawler Network, and outperform all tested competitors with the recommended input parameter values.

The test results show that the system utilization is stabilized, and kept at a higher level compared with the other tested algorithms. However, the results show that the relation between system utilization and fair load-balancing is not that evident. Greedy algorithms,

like IPMicra, may obtain reasonably good system utilization, even if they load-balance the Web-Crawler Network unfair. Further, the KALMAN-BLAWLB parameter variations, have a smaller impact on the system utilization. The only distinct result, is from the experiment that have an higher degree of experimentation. This experiment is able to obtain a higher, but unstable system utilization.

KALMAN-BLAWLB seem to stabilize both the fair load-balancing and system utilization in both the Large[1] and Small[2] environment. However, the time to obtain a stabilized fair load-balancing , i.e. , the equilibrium, is increased when comparing the results from the Large and Small environment. This is probably caused by an increased interval between the decision the KALMAN-BLAWLB is performing, to the feedback is received. In addition the problem complexity increases when more Agent-Tasks is added, so we believe the increased learning time is caused by variety of reasons.

The different KALMAN-BLAWLB parameters seem to benefit on a larger environment. This because the amount of transition noise is decreased when we add more Agent-Tasks. The amount of Agent-Tasks-exchanges, is very stable for all experiments conducted in the Large environment. This means that the KALMAN-BLAWLB parameters is handling the scalability issue fairly well, but further tests in larger environments should be conducted.

To summarize our findings, we can state that BLA-Kalman can be utilized to fairly load-balance a non-stationary Web-Crawler Network. In addition we see that it is able to keep the system utilization and load-balancing stabilized over a longer time-period. We also see that the solution is scalable, but additional experiments should be conducted to test in larger environments and a larger amount of Web-Crawler Agents.

9.1 Further work

The method presented in this thesis, has demonstrated its ability to load-balance a non-stationary Web-Crawler Network. But there are also some not covered elements, that we want to suggest as further work. All the suggestions use KALMAN-BLAWLB as a basis for either concrete future work or research.

Reconfiguration/Resilience

We suggest that the first step in an eventual further development of the KALMAN-BLAWLB, is to develop resilience and reconfiguration. When KALMAN-BLAWLB tolerate failures, i.e. , resilience, and is able to reconfigure the Web-Crawler Network when a failure happen, it is ready for tests in a real web-environment. The most interesting

[1] Web-environment with 1000 Agent-Tasks (1 Agent-Task = 1 host)
[2] Web-environment with 100 Agent-Tasks (1 Agent-Task = 1 host)

aspects to observe when a failure happens, is how KALMAN-BLAWLB is able to learn about the changes and how long time this learning take.

Combing allocation schemes with KALMAN-BLAWLB

Research should be conducted to investigate how allocation schemes can be utilized together with KALMAN-BLAWLB. We have in section 8.3 mentioned an example of a combination of KALMAN-BLAWLB and the breadth-first search allocation algorithm. However, it should also be investigated if an allocation algorithm is possible to couple with the KALMAN-BLAWLB to improve fairness. An good candidate is for instance the re-crawling algorithm presented by Granmo et. al [24].

Extended testing

We suggest that the KALMAN-BLAWLB should be tested further in several challenging situations. One is to extend the amount of Web-Crawler Agents and compare the changes in fairness with the already conducted experiments. Further, a long lasting experiment should be conducted in real or very large simulated web-environment.

Other application areas

It should be investigated if there are other load-balancing problems that can be solved by a similar solution to the KALMAN-BLAWLB. As discussed in section 8.3, KALMAN-BLAWLB only utilize time measurements as feedback and is therefore not particular domain specific to Web-Crawlers. Therefore other domains and load-balancing areas should be investigated.

Appendix A

Acronyms

AI Artificial Intelligence

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BLA Bayesian Learning Automata

CAC Call Admission Control

CLT Central Limit Theorem

CPU Central Processing Unit

DSL Digital Subscriber Line

DAG Directed Acyclic Graph

FIFO First In First Out

FTP File Transfer Protocol

GA Genetic Algorithm

HTTP Hyper Text Transfer Protocol

I/O Input and Output

ICMP Internet Control Message Protocol

IP Internet Protocol

LA Learning Automata

MABP Multi-Armed Bandit Problem

MANBP Multi-Armed Normal Bandit Problem

MPLS Multiprotocol Label Switching

NPP Number Partitioning Problem

QOS Quality Of Service

RIR Regional Internet Registrars

RTT Round Trip delay Time

SMP Symmetric MultiProcessing computer

TABB Two-Armed Bernoulli Bandit

TABBP Two-Armed Bernoulli Bandit Problem

URL Unified Resource Locator

WebCrawlerAgent Web-Crawler Agent

WebCrawlerNetwork Web-Crawler Network

WWW World Wide Web

Appendix B

IPMicra algorithm

Algorithm 5 The IPMicra algorithm

Input: u any newly discovered URL
Initialization: threshold = 50 ms
Method:

```
1:  for all any newly discovered URL u do
2:     subnet s = smallestNonUnary(u);
3:     if IsDelegated(s) then
4:        delegate u, s to the same migrating crawler;
5:        next u;
6:     else if sameCompanySubnetDelegated(s.companyName) then
7:        mc = migrating crawler that has the other subnet;
8:        mc.delegate(u, s) //the url and the subnet
9:     else
10:       while s not delegated do
11:          s = s.parent;
12:          if IsDelegated(s) then
13:             mc = the migrating crawler that has subnet s;
14:             time = mc.probe(u);
15:             if time < threshold then
16:                mc.delegate(u, s)
17:             end if
18:          end if
19:          for all child of s until u is delegated do
20:             sch = s.child
21:             if IsDelegated(sch) then
22:                mc = migrating crawler that has subnet sch;
23:                time = mc.probe(u);
24:                if time < threshold then
25:                   mc.delegate(u, s)
26:                end if
27:             end if
28:             if allAvailableCrawlersProbed then
29:                delegate the subnet to the fastest crawler
30:             end if
31:          end for
32:       end while
33:    end if
34: end for
```

Appendix C

KALMAN-BLAWLB Methods

Algorithm 6 The KALMAN-BLAWLB page visit method

Input: ϕ_j the Agent-Task object; a_j the Web-Crawler Agent that want to receive the page
Initialization:
Method: visitWebPage()

1: $g := g(\phi_j, a_j)$
2: $wait(g)$

Algorithm 7 The KALMAN-BLAWLB add Agent-Task to queue sub-routine

Input: Q_{a_j} existing queue,a_r new selected Web-Crawler Agent,a_j current Web-Crawler Agent
Initialization:
Method: addAgentTaskToQueueOrSendToNew()

1: **if** $a_r = a_j$ **then**
2: $\quad Q_{a_j} \cup \{a_r\}$
3: **else**
4: $\quad sendAgentTaskToAgent(\phi, a_r)$
5: **end if**

Appendix D

Measured HTTP requests

Host	μ (ms)	σ (ms)
http : *//venturebeat.com/*	1845.5	657.64
http : *//www.wired.com/blogs/*	1086.36	305.52
http : *//www.zdnet.com/*	986.34	255.24
http : *//www.slashgear.com/*	713.18	181.23
http : *//blog.rightmobilephone.co.uk/*	1117.7	269.89
http : *//www.crunchgear.com/*	839.62	199.72
http : *//www.geekologie.com/*	955.05	309.66
http : *//jkontherun.com/*	1808.97	431.05
http : *//www.mobileburn.com/*	1768.94	1018.49
http : *//www.intomobile.com/*	662.58	198.63
http : *//www.mobilecrunch.com/*	775.34	221.26
http : *//www.techcrunchit.com/*	1178.44	255.03
http : *//www.electricpig.co.uk/*	2056.89	167.98
http : *//www.engadgetmobile.com/*	763.93	194.49
http : *//arstechnica.com/*	2007.18	1224.59
http : *//www.unwiredview.com/*	1705.08	211.02
http : *//gizmodo.com/*	1999.67	201.39
http : *//www.shinyshiny.tv/*	964.88	642.19
http : *//slashdot.org/*	2397.33	425.13
http : *//www.mobilejaw.com/*	1099.53	273.7
http : *//www.mobileindustryreview.com/*	1454.34	216.76
http : *//www.wired.co.uk/*	1155.6	264.61
http : *//androidcommunity.com/*	1264.66	600.48
http : *//blogspot.com*	793.25	194.76
http : *//www.techcrunch.com/*	1729.4	325.72
http : *//uk.techcrunch.com/*	1729.59	207.74
http : *//www.boygeniusreport.com/*	615.02	254.97
http : *//www.boingboing.net/*	898.45	206.45

$http://mashable.com/$	555.28	237.44
$http://lifehacker.com/$	2150.51	291.67
$http://sethgodin.typepad.com/seths_blog/$	1194.84	309.82
$http://www.typepad.com/$	1426.95	566.83
$http://blogs.telegraph.co.uk/$	2196.54	197.55
$http://www.dailymail.co.uk/$	1404.95	202.55
$http://www.pocket-lint.com/$	2693.96	1052.8
$http://www.guardian.co.uk/tone/blog$	2132.37	262.19
$http://www.readwriteweb.com/$	765.68	208.07
$http://www.engadgethd.com/$	1323.17	987.34
$http://www.timesonline.co.uk/$	2066.97	974.28
$http://consumerist.com/$	2513.45	873.58
$http://www.fonehome.co.uk/$	1199.59	206.35
$http://news.cnet.com/$	1077.56	255.42
$http://scobleizer.com/$	614.56	241.05
$http://seekingalpha.com/$	1192.43	254.71
$http://www.mobileeurope.co.uk/$	1874.57	434.23
$http://blogs.nytimes.com/$	1381.47	183.4
$http://chris.pirillo.com/$	1955.11	456.48
$http://gigaom.com/$	747.95	152.62
$http://techdigest.tv/$	1729.89	877.13
$http://www.techradar.com/blogs$	824.99	218.54
$http://moconews.net/$	1998.7	883.61
$http://www.slashphone.com/$	1121.01	212.99
$http://www.engadget.com/$	563.66	222.65
$http://www.geek.com/$	2226.67	1063.81
$http://www.theregister.co.uk/$	1775.73	202.32
$http://www.buzzmachine.com/$	1376.13	203.05
$http://www.ft.com/$	2444.91	826.75
$http://www.huffingtonpost.com/$	1735.27	206.35
$http://thenextweb.com/$	2955	243.14
$http://www.mattcutts.com/blog/$	1316.5	858.72
$http://www.thisislocallondon.co.uk/$	2101.28	1098.79
$http://www.mobilewhack.com/$	773.23	195.23
$http://www.chrisbrogan.com/$	1124.79	302.43
$http://blogs.extranet.sonyericsson.com/$	2248.88	202.59
$http://blogs.chron.com/techblog/$	1508.56	403.64
$http://www.igizmo.co.uk/blog$	991.93	888.38
$http://www.neowin.net/$	2499.08	225.77
$http://www.gearlive.com/$	1434.34	348.71
$http://www.smartphonedaily.co.uk/$	1963.28	203.59
$http://www.tgdaily.com/$	755.88	145.55

$http://www.blogherald.com/$	756.84	236.56
$http://blackberrysync.com/$	902.54	204.24
$http://dvice.com/$	2445.87	1198.02
$http://www.shortlist.com/$	2055.32	453.15
$http://blogs.independent.ie/$	1114.01	335.46
$http://www.theinquirer.net/$	1106.56	303.17
$http://www.micropersuasion.com/$	1395.67	177.58
$http://www.mobilerule.org/blog/$	1657	1064.64
$http://jeremy.zawodny.com/blog/$	1389.05	209.06
$http://www.livecrunch.com/$	2134.3	1149.86
$http://www.realtechnews.com/$	781.82	173.17
$http://www.gizmag.com/$	2681.52	1092.39
$http://wirelessfederation.com/news/$	791.35	248.61
$http://www.steverubel.com/$	2155.19	668.56
$http://www.allaboutsymbian.com/$	1376.28	199.72
$http://www.lockergnome.com/$	1165.69	336.89
$http://blog.gsmarena.com/$	2740.14	925.67
$http://www.economist.com/blogs/$	1993.59	909.75
$http://www.copyblogger.com/$	1835.53	918.5
$http://www.bbc.co.uk/blogs/$	1447.92	706.56
$http://www.mobilewire.co.uk/mobile/blog$	1623.97	512.01
$http://radar.oreilly.com/$	1147.85	362.34
$http://www.businessweek.com/$	1932.56	986.89
$http://www.marketingweek.co.uk/$	1917.21	209.78
$http://www.broadband-expert.co.uk/blog/$	741.68	201.59
$http://www.telconews.com/$	465.99	216.84
$http://www.phonemag.com/$	734.39	204.02
$http://community.prweek.com$	1605.99	191.83
$http://www.nevillehobson.com/$	2070	732.59
$http://www.tracyandmatt.co.uk/blogs/$	2292.82	1024.31

Appendix E

BLA-Kalman μ convergence

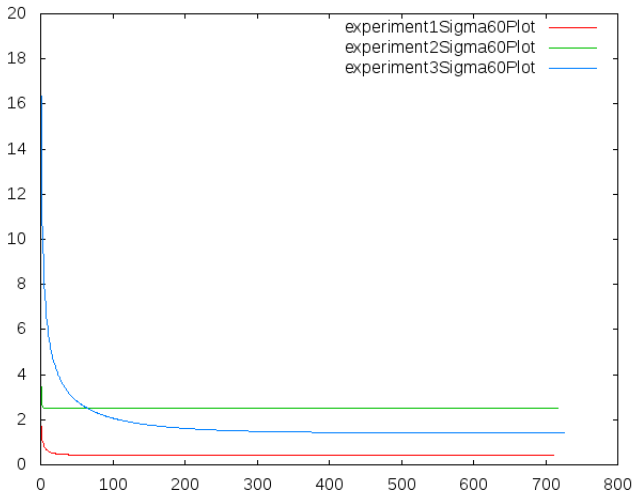


Figure E.1: A plot showing the convergence of $\sigma[N]^2$, using 60 as initial sigma. ($\sigma_{ob} = 4$ $\sigma_{tr} = 2$, $\sigma_{ob} = 20$ $\sigma_{tr} = 0.1$,$\sigma_{ob} = 2$ $\sigma_{tr} = 0.1$))

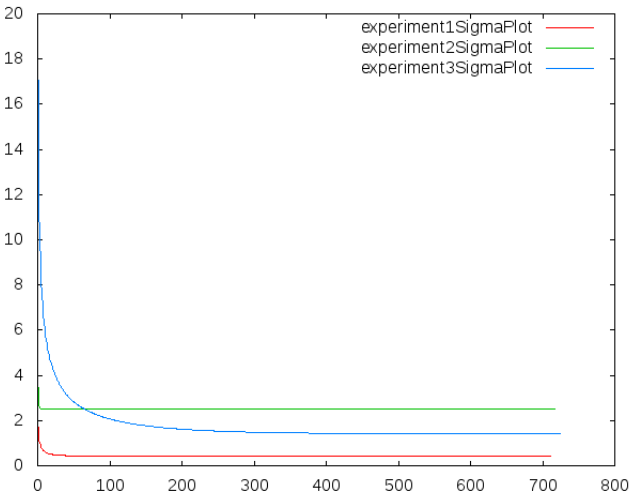


Figure E.2: A plot showing the convergence of $\sigma[N]^2$ using 1000 as initial sigma. ($\sigma_{ob} = 4$ $\sigma_{tr} = 2$, $\sigma_{ob} = 20$ $\sigma_{tr} = 0.1$,$\sigma_{ob} = 2$ $\sigma_{tr} = 0.1$)

Appendix F

Non-interpolated curves

F.1 Fair load-balancing plots

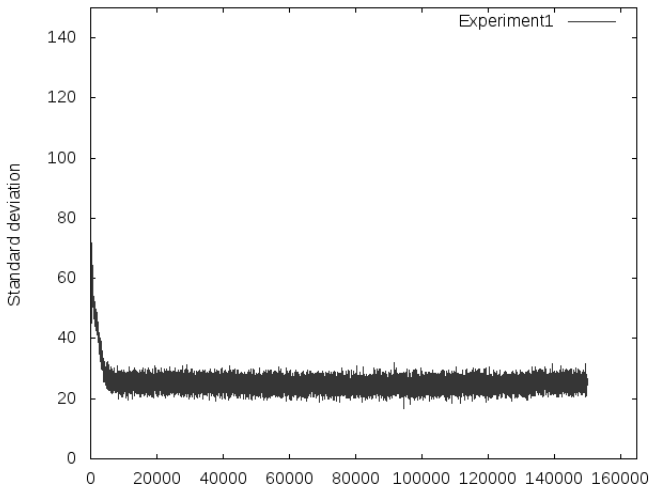


Figure F.1: A non-interpolated fair load-balancing plot of Experiment1. ($\sigma_{tr} = 0.1$ $\sigma_{ob} = 2.0$)

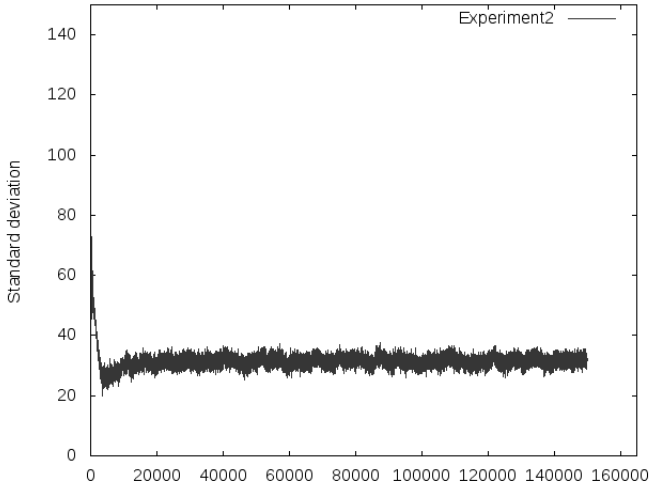


Figure F.2: A non-interpolated fair load-balancing plot of Experiment2. ($\sigma_{tr} = 2.0$ $\sigma_{ob} = 4.0$)

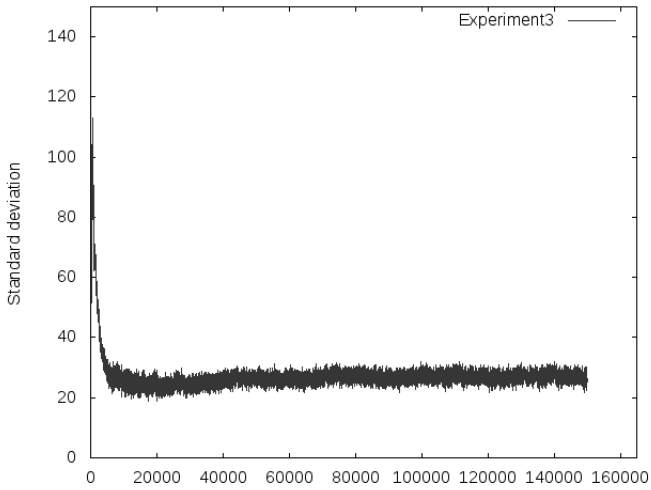


Figure F.3: A non-interpolated fair load-balancing plot of Experiment3.($\sigma_{tr} = 1.0$ $\sigma_{ob} = 20$)

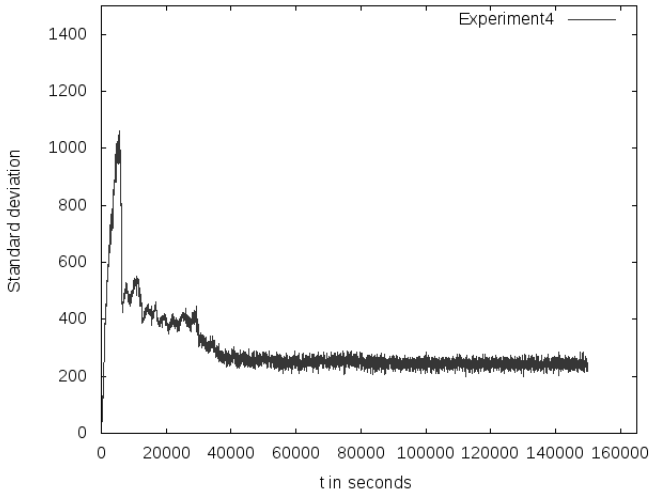


Figure F.4: A non-interpolated fair load-balancing plot of Experiment4.($\sigma_{tr} = 0.1$ $\sigma_{ob} = 2.0$)

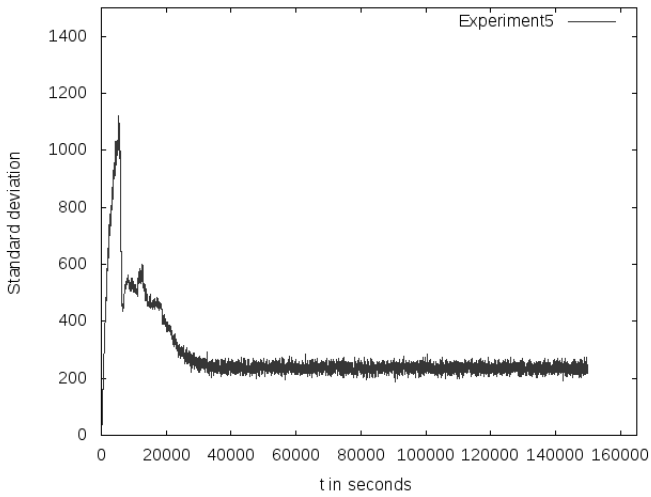


Figure F.5: A non-interpolated fair load-balancing plot of Experiment5. $\sigma_{tr} = 2.0$ $\sigma_{ob} = 4.0$

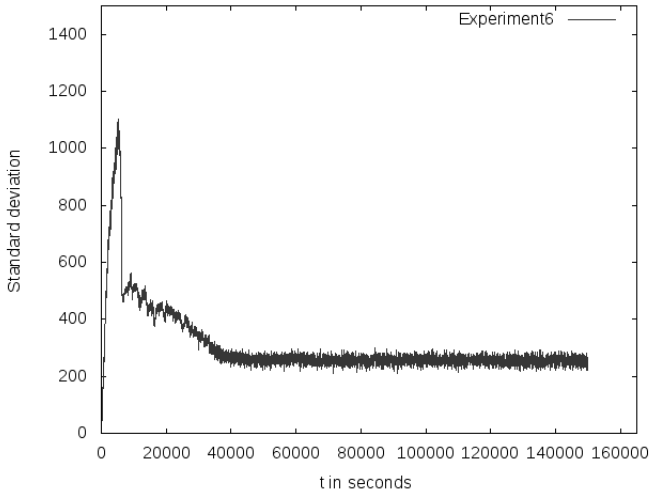


Figure F.6: A non-interpolated fair load-balancing plot of Experiment6.($\sigma_{tr} = 1.0$ $\sigma_{ob} = 20$)

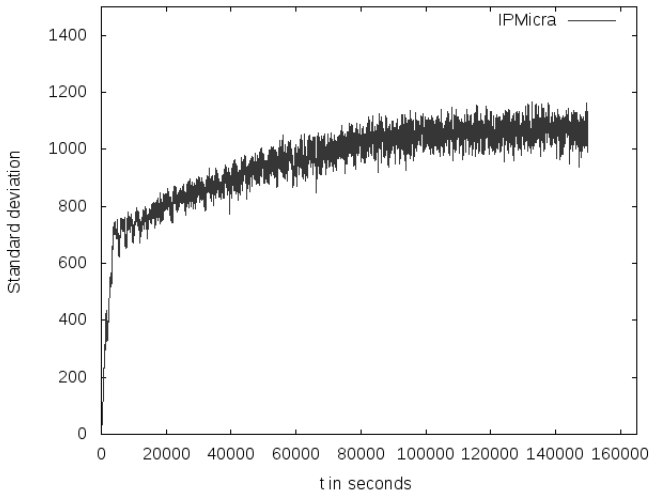


Figure F.7: A non-interpolated fair load-balancing plot of IPMicra in the Large environment

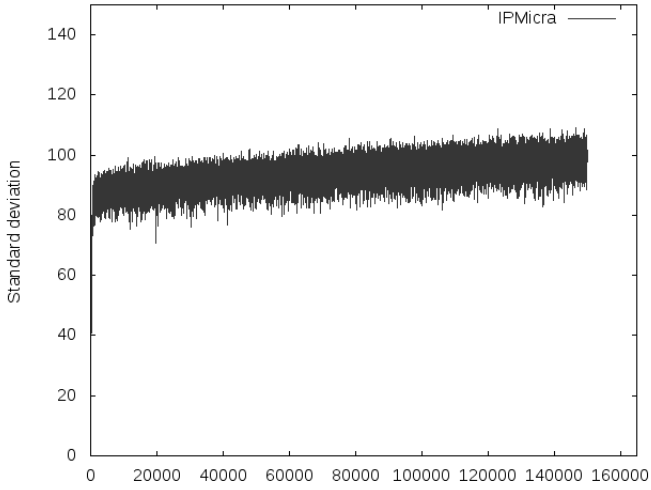


Figure F.8: A non-interpolated fair load-balancing plot of IPMicra in the Small environment

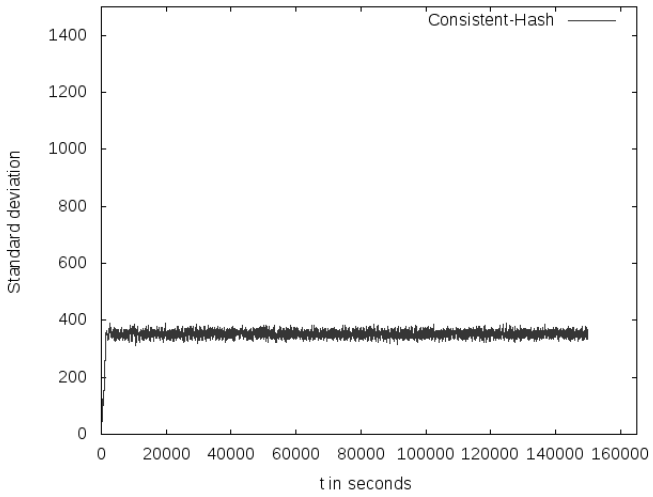


Figure F.9: A non-interpolated fair load-balancing plot of Consistent-Hash algorithm in the Large environment

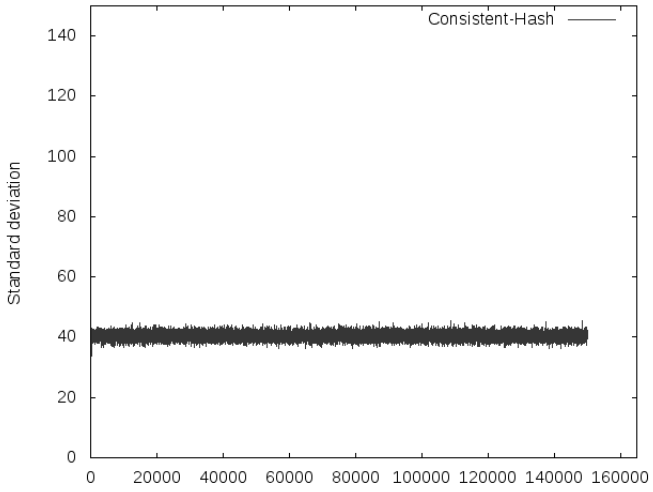


Figure F.10: A non-interpolated fair load-balancing plot of Consistent-Hash algorithm in the Small environment

F.2 System utilization plots

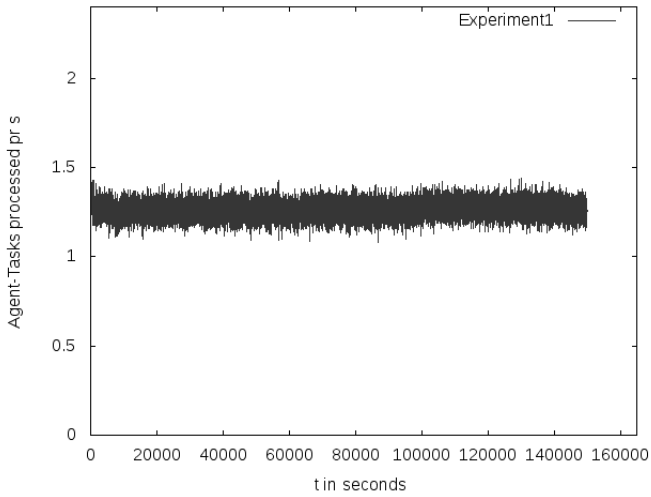


Figure F.11: A non-interpolated system utilization plot of Experiment1.($\sigma_{tr} = 0.1$ $\sigma_{ob} = 2.0$)

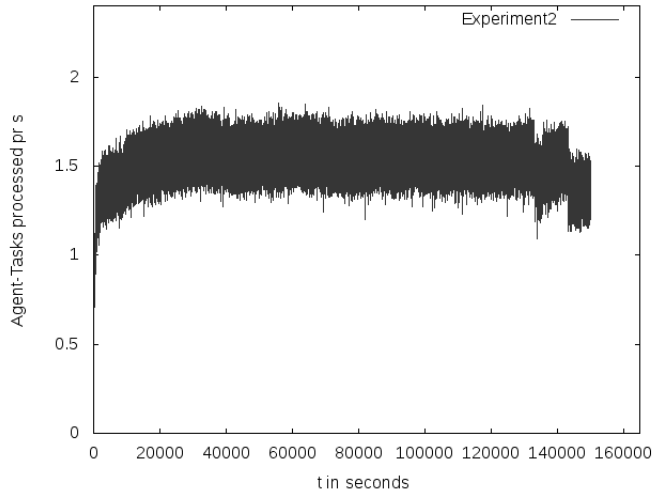


Figure F.12: A non-interpolated system utilization plot of Experiment2. $\sigma_{tr} = 2.0$ $\sigma_{ob} = 4.0$

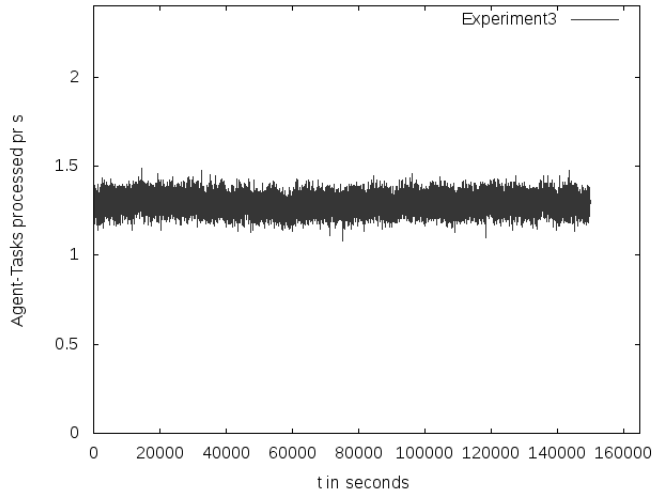


Figure F.13: A non-interpolated system utilization plot of Experiment3.($\sigma_{tr} = 1.0$ $\sigma_{ob} = 20$)

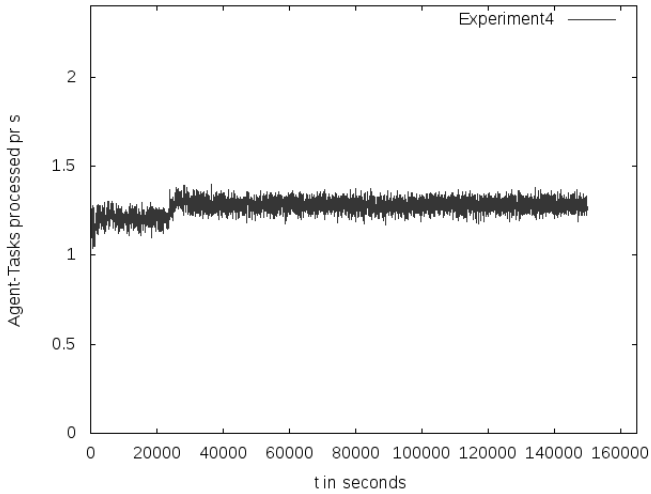


Figure F.14: A non-interpolated system utilization plot of Experiment4.($\sigma_{tr} = 0.1$ $\sigma_{ob} = 2.0$)

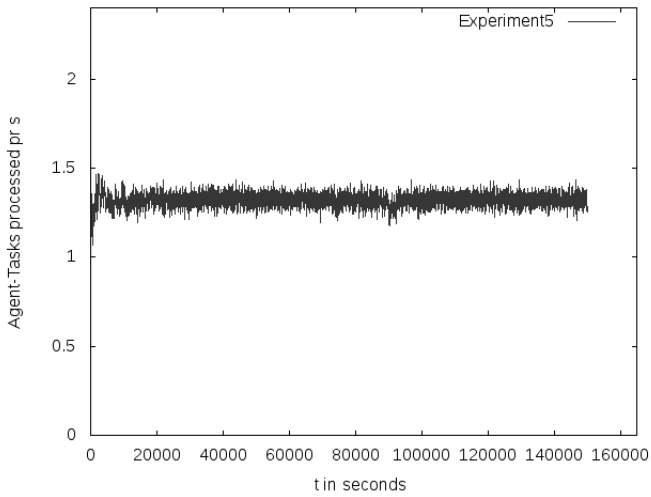


Figure F.15: A non-interpolated system utilization plot of Experiment5.$\sigma_{tr} = 2.0$ $\sigma_{ob} = 4.0$

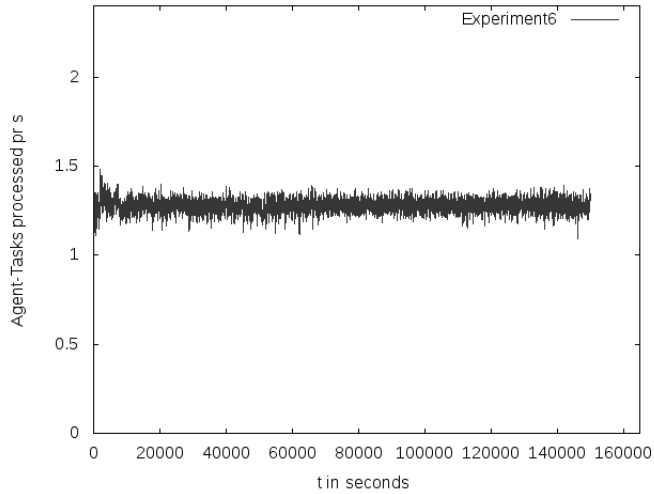


Figure F.16: A non-interpolated system utilization plot of Experiment6.($\sigma_{tr} = 1.0$ $\sigma_{ob} = 20$)

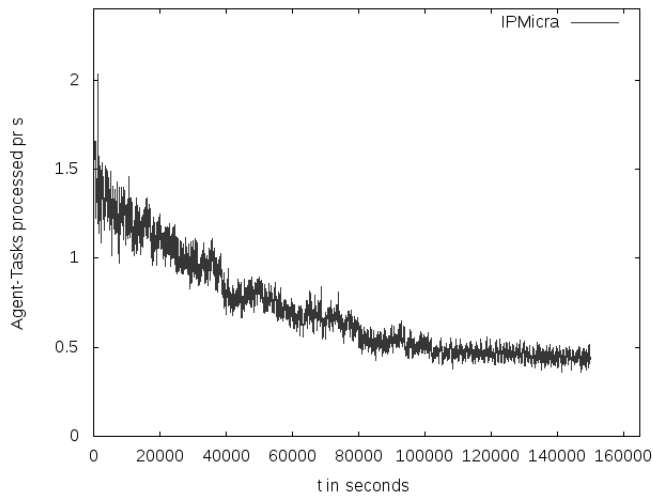


Figure F.17: A non-interpolated system utilization plot of IPMicra in the Large environment

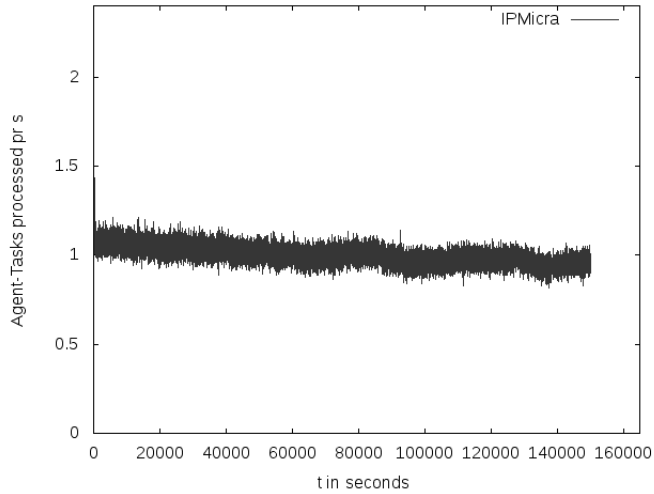


Figure F.18: A non-interpolated system utilization plot of IPMicra in the Small environment

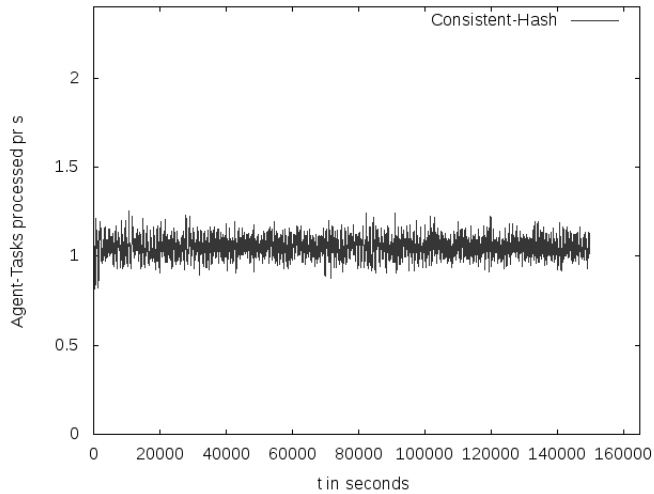


Figure F.19: A non-interpolated system utilization plot of Consistent-Hash algorithm in the Large environment

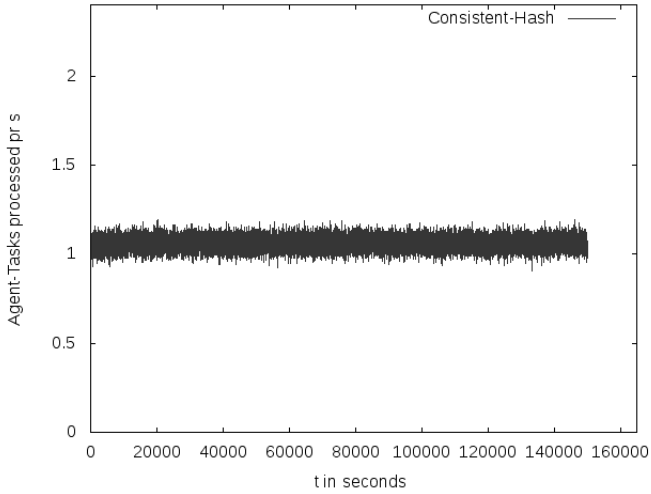


Figure F.20: A non-interpolated system utilization plot of Consistent-Hash algorithm in the Small environment

Appendix G

Plotted experiment settings

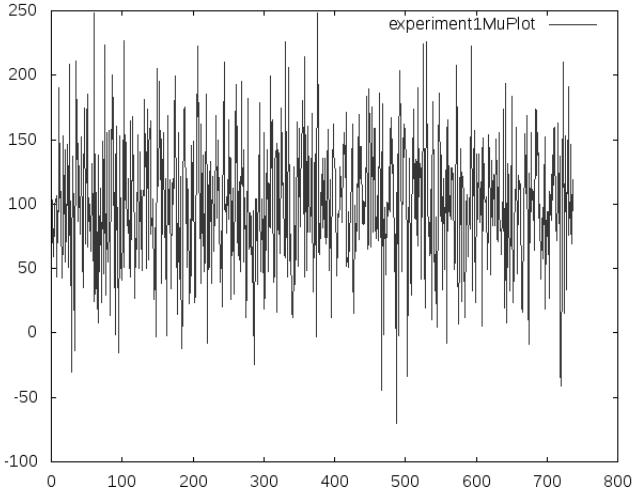


Figure G.1: A plot showing $\mu[N]$ when pulling from a random normal distributed value with $\mu = 100$ and $\sigma = 50$. ($\sigma_{ob} = 2\ \sigma_{tr} = 0.1$)

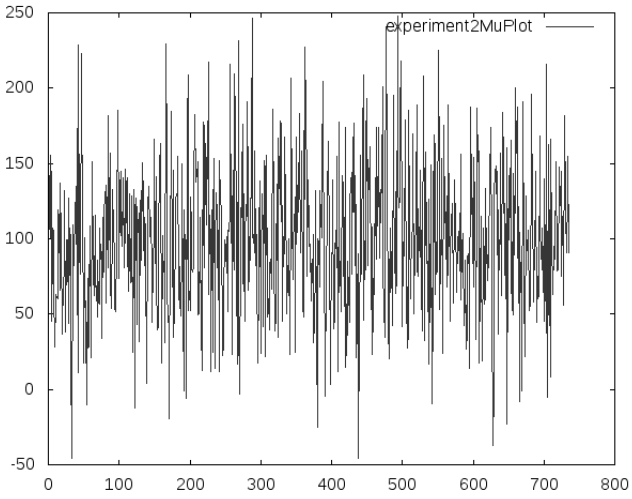


Figure G.2: A plot showing $\mu[N]$ when pulling from a random normal distributed value with $\mu = 100$ and $\sigma = 50$. ($\sigma_{ob} = 4$ $\sigma_{tr} = 2$)

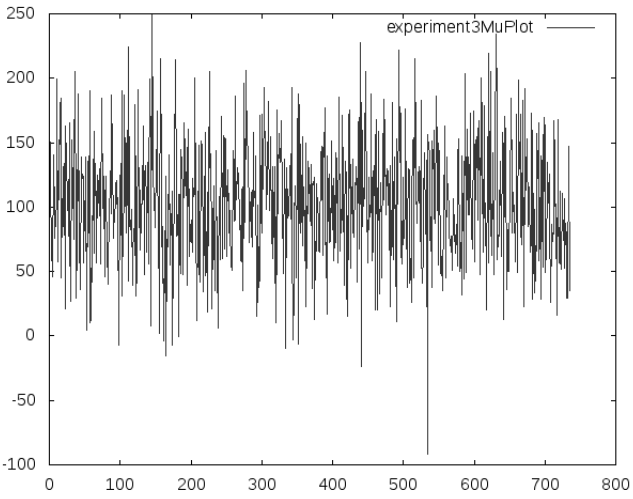


Figure G.3: A plot showing $\mu[N]$ when pulling from a random normal distributed value with $\mu = 100$ and $\sigma = 50$. ($\sigma_{ob} = 20$ $\sigma_{tr} = 0.1$)

Bibliography

[1] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids," in *IEEE Internationa Conference on Advanced Comuting and Communications*, 2000, pp. 45–52.

[2] (2010, May) Apache commons math: Data generation. Apache Fundation. [Online]. Available: http://commons.apache.org/math/userguide/random.html

[3] Y. A. B. Hamidzadeh, D. J. Lilja, "Dynamic scheduling techniques for heterogeneous computing systems," *Concurrency: Practice and Experience*, vol. 7, no. 7, pp. 633–652, 1995. [Online]. Available: http://citeseer.ist.psu.edu/hamidzadeh95dynamic.html

[4] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri, "Challenges on distributed web retrieval," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, April 2007, pp. 6–20.

[5] H. Bauke, S. Mertens, and A. Engel, "Phase transition in multiprocessor scheduling," *Phys. Rev. Lett.*, vol. 90, no. 15, p. 158701, Apr 2003.

[6] K. M. K. Behrooz A. Shirazi, Ali R. Hurson, "Introduction to scheduling and load-balancing," in *Scheduling and load balancing in parallel and distributedsystems*. Los Alamitos, Calif.: IEEE Computer Society Press, 1995, pp. 1–4.

[7] (2010, January) Open-source software for volunteer computing and grid computing. Berkeley EDU. [Online]. Available: http://boinc.berkeley.edu/

[8] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," 2002. [Online]. Available: http://citeseer.ist.psu.edu/boldi02ubicrawler.html

[9] C. Borgs, J. T. Chayes, S. Mertens, and B. Pittel, "Phase diagram for the constrained integer partitioning problem," *Random Struct. Algorithms*, vol. 24, no. 3, pp. 315–380, 2004.

[10] S. Brin, "The anatomy of a large-scale hypertextual web search engine," in *Computer Networks and ISDN Systems*, vol. 30, 1998, pp. 107–117. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.4049

[11] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," *Comput. Netw.*, vol. 33, no. 1-6, pp. 309–320, 2000.

[12] T. Brådland and T. Norheim, "Empirical evaluation of the bayesian learning automaton family," 2009.

[13] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141–154, 1988.

[14] C. Castillo, "Effective web crawling," *SIGIR Forum*, vol. 39, no. 1, pp. 55–56, 2005.

[15] J. Cho and H. Garcia-Molina, "Parallel crawlers," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*. New York, NY, USA: ACM, 2002, pp. 124–135.

[16] (2010, January) comscore releases january 2010 u.s. search engine rankings. ComScore. [Online]. Available: http://www.comscore.com/Press_Events/Press_Releases/

[17] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," pp. 137–150. [Online]. Available: http://www.usenix.org/events/osdi04/tech/dean.html

[18] J. Exposto, J. Macedo, A. Pina, A. Alves, and J. Rufino, "Geographical partition for distributed web crawling," in *GIR '05: Proceedings of the 2005 workshop on Geographic information retrieval*. New York, NY, USA: ACM, 2005, pp. 55–60.

[19] ——, "Geographical partition for distributed web crawling," in *GIR '05: Proceedings of the 2005 workshop on Geographic information retrieval*. New York, NY, USA: ACM, 2005, pp. 55–60.

[20] S. G. W. C. H. D. A. W. M. B. T. C. A. F. R. E. G. Fay Chang, Jeffrey Dean, "Bigtable: A distributed storage system for structured data," 2006.

[21] (2010, January) comscore releases january 2010 u.s. search engine rankings. Forbes. [Online]. Available: http://www.forbes.com/lists/2009/18/global09_The-Global2000_Prof.html

[22] O.-C. Granmo, "A bayesian learning automaton for solving two-armed bernoulli bandit problems," in *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, Dec. 2008, pp. 23–30.

[23] O. C. Granmo and S. Berg, "Solving Non-Stationary bandit problems by random sampling from sibling kalman filters," in *Proceedings of The 23rd International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems IEA-AIE 2010*. Springer-Verlag, Jun. 2010, accepted January 2010.

[24] O.-C. Granmo, B. Oommen, S. Myrer, and M. Olsen, "Determining optimal polling frequency using a learning automata-based solution to the fractional knapsack problem," in *Cybernetics and Intelligent Systems, 2006 IEEE Conference on*, June 2006, pp. 1–7.

[25] (2010, January) Grub. Grub. [Online]. Available: http://www.grub.org/

[26] V. Gupta and R. Campbell, "Internet search engine freshness by web server help," in *SAINT '01: Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 113.

[27] Y. Hongfei, W. Jianyong, and L. Xiaoming, "A dynamic reconfiguration model for a distributed web crawling system," in *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference on*, 2001, pp. 157–162.

[28] ——, "A dynamic reconfiguration model for a distributed web crawling system," in *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference on*, 2001, pp. 157–162.

[29] E. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 2, pp. 113 –120, feb 1994.

[30] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," DEC-TR-301, Digital Equipment Corporation, Tech. Rep., September 1984. [Online]. Available: http://arxiv.org/abs/cs.NI/9809099

[31] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1997, pp. 654–663.

[32] M. Kobayashi and K. Takeda, "Information retrieval on the web," *ACM Comput. Surv.*, vol. 32, no. 2, pp. 144–173, 2000.

[33] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1384–1397, 1988.

[34] G. P. A. V. P. M. M. D. Dikaiakos, D. Katsaros, "Guest editors introduction: Cloud computing distributed internet computing for it and scientific research," in *IEEE INTERNET COMPUTING*, 09 2009, pp. 10–13.

[35] S. Mertens, "The easiest hard problem: Number partitioning," in *Computational Complexity and Statistical Physics*, A. Percus, G. Istrate, and C. Moore, Eds. New York: Oxford University Press, 2006, pp. 125–139, http://arxiv.org/abs/cond-mat/0302536.

[36] M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, March 2007, pp. 1–8.

[37] K. S. Narendra and M. A. L. Thathachar, "Learning automataÂ¿a survey," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-4, no. 4, pp. 323 –334, july 1974.

[38] M. Nasri and M. Sharifi, "Load balancing using consistent hashing: A real challenge for large scale distributed web crawlers," in *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on*, May 2009, pp. 715–720.

[39] B. J. Oommen, S. Misra, and O.-C. Granmo, "Routing bandwidth-guaranteed paths in mpls traffic engineering: A multiple race track learning approach," *IEEE Trans. Comput.*, vol. 56, no. 7, pp. 959–976, 2007.

[40] O. Papapetrou, S. Papastavrou, and G. Samaras, "Ucymicra: Distributed indexing of the web using migrating crawlers," in *In Proceedings of the 7th East-European Conference on Advanced Databases and Information Systems*, 2003, pp. 133–147.

[41] O. Papapetrou and G. Samaras, "Ipmicra: Toward a distributed and adaptable location aware web crawler," in *ADBIS (Local Proceedings) 2004*, 2004.

[42] ——, "Minimizing the network distance in distributed web crawling," in *CoopIS/DOA/ODBASE*, 2004, pp. 581–596.

[43] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[44] C. A. Penmatsa, S., "Cooperative load balancing for a network of heterogeneous computers," April 2006, p. 8 pp.

[45] B. Pinkerton, "Finding what people want: Experiences with the WebCrawler," in *Proceedings of the 2nd International World Wide Web*, ser. Online & CDROM review: the international journal of, Anonymous, Ed., vol. 18(6). Medford, NJ, USA: Learned Information, 1994.

[46] C. D. Polychronopoulos and D. J. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers," *IEEE Trans. Comput.*, vol. 36, no. 12, pp. 1425–1439, 1987.

[47] G. N. S. Prasanna and B. R. Musicus, "Generalized multiprocessor scheduling for directed acyclic graphs," in *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 237–246.

[48] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," in *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 129–138.

[49] D. E. Repository, "The rbse spider - balancing effective search against web load," 1994.

[50] J. A. Rice, *Mathematical Statistics and Data Analysis*. Duxbury Press, April 2001. [Online]. Available: http://www.worldcat.org/isbn/0534399428

[51] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=773294

[52] (2000, July) Scaling up vs. scaling out. SQLServer Magazine. [Online]. Available: http://www.sqlmag.com/Articles/Index.cfm?ArticleID=8755DisplayTab=Article

[53] (2000, July) Scale-out vs. scale-up: Which technique is right for your data center? processor.com. [Online]. Available: http://processor.com/editorial/article.asp?article=articles/P2913/33p13/33p13.aspguid=

[54] B. Shirazi, M. Wang, and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," *J. Parallel Distrib. Comput.*, vol. 10, no. 3, pp. 222–2232, 1990.

[55] A. S. Tanenbaum and M. V. Steen, *Distributed Systems: Principles and Paradigms*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[56] C. Unsal. (2009, May) Intelligent navigation of autonomous vehicles in an automated highway system: Learning methods and interacting vehicles approach. Digital Library and Archives. [Online]. Available: http://scholar.lib.vt.edu/theses/available/etd-5414132139711101/

[57] S. Zheng, P. Dmitriev, and C. L. Giles, "Graph-based seed selection for web-scale crawlers," in *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*. New York, NY, USA: ACM, 2009, pp. 1967–1970.