Dynamic Fault-Tolerance and Task Scheduling in Distributed Systems

Philip Ståhl
ada10pst@student.lu.se

Jonatan Broberg
elt11jbr@student.lu.se

February 26, 2016

Master's thesis work carried out at Mobile and Pervasive Computing Institute (MAPCI), Lund University.

Supervisors: Björn Landfeldt, bjorn.landfeldt@eit.lth.se

Examiner: Christian Nyberg, christian.nyberg@eit.lth.se

Abstract

This document describes the Master's Thesis format for the theses carried out at the Department of Computer Science, Lund University.

Your abstract should capture, in English, the whole thesis with focus on the problem and solution in 150 words. It should be placed on a separate right-hand page, with an additional *1cm* margin on both left and right. Avoid acronyms, footnotes, and references in the abstract if possible.

Leave a *2cm* vertical space after the abstract and provide a few keywords relevant for your report. Use five to six words, of which at most two should be from the title.

Keywords: MSc, template, report, style, structure

Acknowledgements

If you want to thank people, do it here, on a separate right-hand page. Both the U.S. *acknowledgments* and the British *acknowledgements* spellings are acceptable.

We would like to thank MAPCI and our supervisor Björn Landfeldt for there input. Shub...something for input. We also would like to thank our examiner Christian Nyberg.

Contents

Chapter 1
Introduction

1.1 Background and Motivation

Ensuring a certain level of reliability is of major concern for many vendors. As cloud computing are growing rapidly and users are demanding more services and higher performance load balancing and task scheduling in the cloud has become a very important and interesting research area. Cloud systems often consists of heterogeneous hardware and ensuring a certain level of reliability is therefore a complex task as any of the components may fail at any time.

Guaranteeing properties such as high reliability, raises complexity to the resource allocation decisions, and fault-tolerance mechanisms in highly dynamic and complex distributed systems. For cloud service providers, it is necessary that this increased complexity is taken care of without putting extra burden on the user. The system should therefore ensure these properties in a seamless way.

As computational work is distributed across multiple resources, the overall reliability of the application decreases. To cope with this issue, a fault-tolerant design or error handling mechanism need to be in place. This is of particular interest for cloud service providers, as users often desire a certain level of reliability. In some cases, vendors, for example carrier providers, are even obliged by law to achieve a certain level of availability or reliability. In these cases, one may need to sacrifice other quality aspects such as latency and resource usage. By using static and dynamic analysis of the infrastructure and the mean-time-to-failure for the given resources, a statistical model can be used in order to verify that the desired level is reached.

Reliability can be increased by cloning application tasks, where all clones produce the same output. This allows for both increased redundancy and an easy way of detecting errors. If the execution of one clone stops or the link between a clone and the receiver is broken, this can easily be detected at the receiver side and the scheduler mechanism can automatically find a proper location for a new clone. This allows for continuing execution

of the application even in case of e.g. hardware failure. Seamlessly being able to continue the execution, without losing any data is of particular interest in data stream processing.

Some of the drawbacks of replicating a task n times, is that the resources needed increases. With n replicas, all performing the same computation on the same input, one need n times as much resources, and since only the work of one task is needed, a lot of computational resources is thus wasted. Dynamic analysis of the system could help in determining on which resources one should assign the tasks to for optimal resource usage and load balancing.

1.2 Related work

A lot of work has been done in the area of designing fault-tolerant applications. Most of them use a checkpoint/restart based approach [X][Y][TODO]. These examples relies of the notion of a stable storage, such as a hard-drive, which is persistent even in the case of a system failure.

Previous work has been done which aims at achieving fault-tolerance through replication. [15] aims at providing a certain level of fault-tolerance through replication in a Grid environment, and calculates the number of replicas based on the reliability of the grid resources. However, their approach is limited to the fact that they use static replication, i.e. when a replica fails it is not replaces with a new one. While this may be sufficient for task with short execution times, it will be insufficient for long running tasks, as one or several replicas then are likely to die during the time of execution, and thereby not fulfilling the level of fault-tolerance. Furthermore, the reliability model used is based on non-repairable objects, why they use a Weibull distribution of failures.

maximized reliability under end-to-end delay constraint. Several approaches can be employed to improve the execution reliability such as task duplication and checkpoint-restart. Task duplication executes each task more than once in order to decrease the probabil- ity of failure by redundancy. The main problem of this approach is increased network traffic and computing needs. Alternatively, it is possible to record the checkpoints and restart the application when a failure occurs. However, this approach may also incur extra overhead and slow down the execution by restarting the application and repeated computations and communications [14]. [algoMaxRelEndToEndConstraint]

In this paper, we consider a heterogeneous DS that runs a hard real-time application (all the tasks should meet their deadlines). The goal is to find a task to processor assignment under which the overall reliability of the system (probability that all tasks run successfully) is maximized, while memory, communication bandwidth, and processing load constraints, as well as real-time requirements are met. [11]

To reduce the effect of failures on an application executing on a failure-prone system, matching and scheduling algorithms which minimize not only the execution time but also the probability of failure of the application must be devised [9]

Reliability of distributed simulation is define to be the probability that the given distributed simulation application can run successfully under the distributed computing environment. (assumes execution time) [10]

We present an efficient scheme based on task replication, which utilizes the Weibull reliability function for the Grid resources so as to estimate the number of replicas that

are going to be scheduled in order to guarantee a specific fault tolerance level for the Grid environment. The number of replicas is calculated by the Grid middleware and is based on the failure probability of the Grid resources and the policy that is adopted for providing a specific level of fault tolerance. The adopted replication model is based on static replication [2], meaning that, when a replica fails, it is not substituted by a new one. replicas causes an overhead in the workload that is allocated to the Grid for execution. Scheduling and resource management are important in optimizing Grid resource allocation [3] and determining its ability to deliver the negotiated Quality of Service to all users [4,5] [15]

To optimize the performance of a DCS, several issues arise such as the minimization of time and cost as well as maximization of system reliability [9, 26]. [An Algorithm for Optimized Time, Cost, and Reliability in a Distributed Computing System]

Furthermore, [X] [Y] and [Z] focus not on the reliability that the application provides a result, but that it provides the correct result. This is achieved by various consensus algorithms. [Scheduling Fault-Tolerant Distributed Hard Real-Time Tasks Independently of the Replication Strategies]

Byzantine consensus among component replicas is another form of monitoring. The replicas vote on the correct output or action based on the observed inputs. In the event of a disagreement, the minority is considered faulty. This method allows for the detection of Byzantine faults, resulting in a much more powerful monitor. [4]

Consensus [Fundamentals of Fault-Tolerant Distributed Computing]

Traditional redundancy, also called k-modular redun- dancy [38], which performs k 2 f3; 5; 7; . . .g indepen- dent executions of the same task in parallel and then takes a vote on the correctness of the result. Progressie-redundancy, self-adaptive iterative redundancy [Self-Adapting Reliability in Distributed Software Systems]

Scheduling: Some focus on reliability while ensuring time constraints? [Z][TODO].

SLA aware scheduling: (i) to enable cloud storage systems with I/O performance management (ii) to minimize the I/O throughput SLA violations using effective scheduling policies. I/O throughput SLA in cloud storage service is defined as the I/O throughput of user's volume is higher or equal to a specific number of IO operations per second (IOPS) in at least 99.9% of time. [SLA-aware Resource Scheduling for Cloud Storage]

Previous work usually assumes a constant failure rate [8] [9] [10] [11] [12]. Such assumptions are unlikely to model the dynamicity of heterogeneous systems correctly. Therefore, we adapt a self-learning model, which adapts the failure rate for nodes in the system, and tries to avoid scheduling of tasks on unreliable nodes.

This paper proposes a service-oriented software reliability model that dynamically evaluates the reliability of Web services. There are two kinds of Web services: atomic services without the structural information and the composite services consisting of atomic services. The model first evaluates the reliability of atomic services based on group testing and majority voting. Group testing is the key technique proposed in this paper to support the service-oriented reliability model. This paper proposes a Service-Oriented software Reliability Model (SORM). This model evaluates the reliability of WS in two steps: (1) Use highly efficient group testing to evaluate the reliability of atomic services. (2) Evaluate the reliability of a composite service based on the reliabilities of the component services (they can be atomic services or composite services) and the structure (relationships) among the component services. Both evaluation steps are dynamic and at runtime. [A

SOFTWARE RELIABILITY MODEL FOR WEB SERVICE]

For reliability evaluation, researchers often design some algorithms to simplify Markov models, generate corresponding File Spanning Tree (FST) to evaluate the K-terminal reliability factoring theorems [2, 7, 9, 10] or the graph theories and probability synthetically Symbolic Method (SM) and Factoring Method (FM) algorithms were proposed to compute the reliability of a distributed computing system with imperfect nodes [10]. [A Real-Time Performance Evaluation Model for Distributed Software with Reliability Constrains] [7]

1.3 Our contributions

Most previous work uses statis, non-adaptive, probability models for determining the reliability of a node or a system of nodes. Furthermore, assumptions such as constant failure rate are often made, even though they do not reflect reality very well [X]. Instead, we use an static model only at deployment of a system, after which the model self-adapts to the real situations. As failures are encountered, the mean-time-to-failure is updated.

Furthermore, previous work related to reliability either defines reliability as that the application produces the correct result [X], or they bla bla bla. In our work, we only checkpoint the state of the system, so we do not lose any information about the system in case of a failure.

... calculating number of replication at deployment ... ours is dynamic and the number of replicas changes over time as the reliability of the system varies.

... known execution time ... our application is a long running data stream processing application, with no known execution time, of indefitive execution time. While some previous work focus on the reliability that the application finishes before a node failure, we focus on the reliability that at least one replica is always running.

Chapter 2
Background Theory

2.1 Computational Environment

2.1.1 Types of distributed computing

There are a number of different types of distributed environments, e.g. grid, clusters, cloud and Heterogeneous Distributed Computing System (HDCS).

Grid computing is the collection of autonomous resources that are distributed geographically and by a single domain work together to achieve a common goal, i.e. to solve a single task [3].

Clusters are similar to grid but with the difference that they aren't distributed geographically. They work in parallel under supervision of a single administrative domain. From the outside it looks like a single computing resource. [3]

Cloud can be described as the next generation of grids and clusters. While it is similar to grid and clusters, for example parallel and distributed, the important difference is that cloud has multiple domains [3]. Machines can be geographically distributed, and software and hardware components are often heterogeneous, and analyzing and predicting workload and reliability is usually very challenging [2].

HDCS, is a system of numerous high-performance machines connected in a high-speed network and therefore high-speed processing of heavy applications is achieved.

2.1.2 Dynamic versus static environments

In static environment the cloud provider installs homogeneous resources. Also the resources in the cloud are not exible when environment is made static. In this scenario, the cloud requires prior knowledge of nodes capacity, processing power , memory, performance and statistics of user requirements.

In dynamic environment the cloud provider installs heterogeneous resources. The resources are exible in dynamic environment. In this scenario cloud cannot rely on the prior knowledge whereas it takes into account run-time statistics. [3]

2.2 Faults in distributed environments

2.2.1 Types of Faults

In distributed environments, especially with heteregenous commodity hardware, several types of failures can take place, which may affect the running applications in the environment. These failure include, but are not limited to, hardware malfunction, software errors, network failure, loss of energy, etc.

In [13], real-world failure data is studied and in that data, hardware failures was the single most common type of failure, ranging from 30 to more than 60%.

(i) Crash failures, where a server halts, but was working correctly until that moment; (ii) Omission failures, where a server fails to respond to incoming requests and to send messages; (iii) Timing failures, where the server succeeds in responding but its response is beyond the specified time interval. [15]

2.2.2 Fault life-cycle techniques

prevention, removal, tolerance, forecasting. [2]

2.2.3 Fault models

Byzantine Faults

[4]

in which pro- cessors may behave in arbitrary, even malevolent, ways). System correctness was always proved with respect to a specific fault model [Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments]

Fail-stop (crash-stop?) Faults

[4] fail-stop (in which a processor crashes, but this may be easily detected by its neighbors) [Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments]

This failure model has the practical implication that avoids the use of complicated software and/or hardware replication systems for recovering tasks from failed server. Consequently, the application cannot be successfully serviced by the system if at least one task remains unprocessed at a failed server. [Performance and Reliability of Non-Markovian Heterogeneous Distributed Computing Systems]

Fail-stutter Faults

[4]

Crash-failure model

Well- known examples are the crash failure model (in which processors simply stop executing at a specific point in time), [Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments]

2.2.4 Failure Distribution

Previous work usually assumes resource failure within a system follows a Poisson process with a constant failure rate [8] [9] [10] [11] [12]. For this model to be valid, it is assumed that failures between resources are statistically independent [8].

$$MTBF = (nbroffailures)/(totaltime) \tag{2.1}$$

Assumptions such as constant resource failure rate and statistically independent failures, are not likely to model the actual failure scenario of a dynamic heteregenous distributed system [9]. [2] show that the probability of hardware failure are greater in the beginning and end of lifetime. Furthermore, by studying failures in high-performance computing systems, [13] found a Poisson distribution to be a poor fit, while a normal or lognofmrla distribution fitted the data much better.

Failure assumptions

possible assumptions:

- Each component of the distributed system (node, communication link, LP being executed) only has two states: operational or failed.
- Failures of components are statistically independent.
- The “future life” of a component (the time to the next failure) follows exponential distribution, i.e., the failure of a component follows a Poisson process.

2.3 Fault tolerance techniques

Fault-tolerance techniques can be divided into two categorizes: Task-level techniques and work-flow level techniques. Task-level techniques refer to those techniques which are applied in task-level to mask the effect of task crash failures. Work-flow techniques are those techniques that basically allow alternative task to launch to deal with user-defined exceptions and the failures the task-level techniques fail to cover [5].

Fault tolerance techniques can also be divided into reactive and proactive techniques. A reactive fault tolerant technique reacts when a failure occur and tries to reduce the effect of the failure. The proactive technique on the other hand tries to predict failures and proactively replace the erroneous components.

One way of taking failures into account is to employ a reliable matching and scheduling algorithm in which tasks of an application are assigned to machines to minimize the probability of failure of that application. This type of approach was followed for allocating

undirected task graphs to nonredundant and redundant real-time distributed systems [8], [9], [10], [11].

The basics, however, are checkpoint recovery and task replication The former is a common method for ensuring the progress of a long-running application by taking a checkpoint, i.e. saving its state on permanent storage periodically. A checkpoint recovery is an insurance policy against failures. In the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be re-computed. Task replication is another common method that aims to provide fault tolerance in distributed environments by scheduling redundant copies of the tasks, so as to increase the probability of having at least a simple task executed [15].

2.3.1 Checkpoint/Restart

The basics, however, are checkpoint recovery and task replication. The former is a common method for ensuring the progress of a long-running application by taking a checkpoint, i.e. saving its state on permanent storage periodically. A checkpoint recovery is an insurance policy against failures. In the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be re-computed [15]

2.3.2 Replication

There are a number of different strategies for replicating a task:

- Active - A replica is run on a second machine and receives an exact copy of the primary nodes input. From the input it performs the same calculations as if it was the primary node. It monitor the primary node for incorrect behavior and in event that the primary node fails or behaves in an unexpected way the replica will promote itself as the primary node [4]. This type of replication is feasible only if by assumption that the two nodes receives exactly the same input. Since the replica already is in an identical state as the primary node the transition will take negligible amount of time. A drawback with active replication is that all calculations are ran twice, thus a waste of computational capacity.
- Semi-Acitve - Semi-active replication is very similar to active replication but with the difference that decisions common for all replicas are taken by one site.
- Passive - A second machine, typically in idle or power off state has copy of all necessary system software as the primary machine. If the primary machine fails the "spare" machine takes over, which might incur some interrupt of service. This type of replication is only suitable for components that has a minimal internal state, unless additional checkpointing is employed.

The above corresponds to the probability of the event “all the replicas and the original task fail”. Respectively, the success probability is equal to the probability of the event “the original task or at least one of its replicas executes successfully”. The number of replicas issued depends on the failure probabilities of the original tasks and on the desired fault tolerance level in the Grid infrastructure.

2.4 Reliability Model

I order to examine a systems reliability we must first define what we mean with reliability. One definition is that reliability is the probability that the system can run an entire task successfully [1]. A similar definition is that reliability is the ability of a software application that in a certain environment maintain a performance level for a specified amount of time. Reliability could also be interpreted more as the availability of a system, for example a percentage describing the up-time of the system. [2] defines a software application reliable if the following is achieved:

- Perform well in specified time t without undergoing halting state
- Perform exactly the way it is designed
- Resist various failures and recover in case of any failure that occurs during system execution without proceeding any incorrect result.
- Successfully run software operation or its intended functions for a specified period of time in a specified environment.
- Have probability that a functional unit will perform its required function for a specified interval under stated conditions.
- Have the ability to run correctly even after scaling is done with reference to some aspects.

For simplicity we will start with a very simple reliability definition, Definition 2.1.

Definition 2.1. *The reliability of a process is measured by the probability that the hardware is functioning during the execution time.*

Later on we use a more dynamic definition, Definition 2.2 which is a combination of the above definitions.

Definition 2.2. *The reliability of a process is measured by the probability that the task is executed within a specified amount of time without experiencing any type of failure (internal or external) during it's time of execution.*

In the case of stream-data processing, we assume a continously running application, without a deterministic execution time. Instead, a more suitable definition of reliability in this case is:

Definition 2.3. *The reliability of a process with several replicas is defined as the probability at least one replica is always running. This can be expressed as the probability that not all replicas fail during the time it takes start a new replica of the first dying one.*

To fully understand the above definitions we give a brief definition of the three terms failure, fault and error.

- Failure: Is defined as the event, e.g. when the software produces the wrong result or when a server crashes. Or simply the inability to perform as required.

- Fault: A fault is the state of software. It is also called defect/bug.

- Error: An error is a human mistake.

To give a clear picture of the definitions above you can say that when a human writes code and does a mistake an error is committed and a fault is introduced in the code. Later, when the fault effects the execution of the software in an undesired way we have experienced a failure.

Since many real-time systems are non-deterministic environment in which even dangerous, system fault tolerance is necessary and important. In order to effectively verify the level of fault-tolerant systems, a reliability model needs to be proposed, it is generally assumed that the error arrival rate is stable and the error probability at any time intervals meets Poisson distribution [7-8] [**?**]

While redundancy and diversity have been employed as popular methods to attain better reliability [4][5][6][7][8][9][10], they impose extra hardware or software costs. optimal task allocation. This method does not require additional hardware or software and improves system reliability just by using proper allocation of the tasks among the nodes [8][11][12] [13] [14] [11]

the reliability of processing node Pk in time interval [0,t] can be achieved by: R (t)= (5) P where is hazard rate at time . By assuming a constant hazard rate (like [3] and [25]), Eq. (5) reduces to: RP (t) = t (6) Total hazard rate of each node can be computed by adding failure rate of its hardware and failure rate of its OS if we assume that they are independent

Discrete and continuous reliability models for systems with identically distributed correlated components.pdf

System reliability: Due to independence of node and path failures, system reliability can be formulated as: Rs(X) = RS'(X) * Rs" (X) [11]

the reliability model of the subdistributed systems inherits the traditional models' characters [19], [20], [21], [22] and has certain limitations. Those traditional models have a com- mon assumption that the operational probabilities of the nodes and links are constant. However, this assumption is unrealistic for the grid, so this assumption was relaxed in [15] by assuming that the failures of nodes and links followed their respective Poisson processes so that their operational probabilities decrease with their working time instead of the constant values There are also many other reliability models for software, hardware, or small-scale distributed systems, see, e.g., [14], [23], [24], [25], [26], which cannot be directly implemented for studying grid service reliability. [A Hierarchical Modeling and Analysis for Grid Service Reliability.pdf]

Reliability evaluation of the hypercubes has been addressed recently [7]-[9]. Najjar and Gaudiot have modeled hypercube reliability assuming that the system works as long as there is no disconnectedworking node(s) [7]. This implies that even if a task requirement is satisfied, the system is considered failed whenever there is a disconnected node(s). This reliability is not based on task requirement, but on system managemen. [A unified task-based dependability model for hypercube computers]

The dependability model is based on the following assump- tions All the processors (nodes) are homogeneous with iden- tical and exponential distribution of failure time. We define X as the failure rate of a node. Fortheavailabilitymodel,thereisasinglerepairfacility with exponential distribution of repair time. The average repair rate is given by p. The

link failure rate is negligible compared to the node failure rate. [A unified task-based dependability model for hypercube computers]

(the processing element that runs the program under consideration) to some other nodes such that its vertices hold all the needed files for the program under consideration. [**?**]

This paper proposes a service-oriented software reliability model that dynamically evaluates the reliability of Web services. There are two kinds of Web services: atomic services without the structural information and the composite services consisting of atomic services. The model first evaluates the reliability of atomic services based on group testing and majority voting. Group testing is the key technique proposed in this paper to support the service-oriented reliability model [A SOFTWARE RELIABILITY MODEL FOR WEB SERVICE]

What affects software reliability? ...lists 32 factors ... [An analysis of factors affecting software reliability]

The most common simplifying assumptions em- ployed in the literature are heuristics related to applications' service time [7], [8], [9], homogeneous capabilities of servers and/or communication links [8], time-invariant DCS topol- ogy [10], and the deterministic behavior of the transfer time of tasks [9], [11], [12]. [Performance and Reliability of Non-Markovian Heterogeneous Distributed Computing Systems]

In this paper, we consider three key performance and reliability metrics: the average service time of an application, the QoS and the service reliability in executing an application [2], [3], [16], [22]. While precise mathematical definitions of these metrics are presented in Section 3.3, at this point we will briefly introduce them and describe their scope of applicability. The average service time is critical to assess the speedup in the runtime of applications when executed in parallel on a DCS. The average service time is a reasonable metric (i.e., it takes a finite value) only in settings where servers are completely reliable or in settings where servers are allowed to recover after a failure. The QoS, which is defined as the probability of executing an application by a prescribed time deadline, is a reasonable metric in settings where server nodes may or may not fail. The QoS metric is of interest to system users and analysts, specially in real-time or in time constrained applications. Finally, the service reliability, which is defined as the probability of successfully executing an entire application, is an important metric for assessing the dependability of applications executed on DCSs, and it is a reasonable metric only when servers can fail without recovery and/or in settings where applications cannot continue their execution after a failure [Performance and Reliability of Non-Markovian Heterogeneous Distributed Computing Systems]

We focus on the study of task allocation deci- sion to achieve maximal distributed system reliability (DSR) under processor resource constraints and total system cost constraint. the task allocation problem has been studied to achieve various goals, such as reliabil- ity maximization, safety maximization, fault tolerance increasing, and cost minimization. Their processor reliability was com- puted from the processor failure rate and the elapsed execution time. As far as we know, the effects of module software reliabilities and module execution frequencies on the optimal task allocation decision have not been studied before. To pursue this topic, we di- vide the processor reliability into two parts in our system model: the processor hardware reliability and the module software reliability [The decision model of task allocation for constrained stochastic distributed systems]

2.5 Load balancing

The term load balancing is generally used for the process of transferring load from over-loaded nodes to under loaded nodes and thus improving the overall performance. Load balancing techniques for a distributed environment must take two tasks into account, one part is the resource allocation and the other is the task scheduling.

The load balancing algorithms can be divided into three categorize based on the initiation of the process:

- Sender Initiated - An overloaded node send requests until it find a proper node which can except its load.
- Receiver Initiated - An under loaded node sends a message for requests until it finds an overloaded node.
- Symmetric - A combination of sender initiated and receiver initiated.

Load balancing is often divided into two categorize, namely static and dynamic algorithms. The difference is that the dynamic algorithm takes into account the nodes previous states and performance whilst the static doesn't. The static load balancing simply looks at things like processing power and available memory which might lead to the disadvantage that the selected node gets overloaded [6]

A dynamic load balancing can work in two ways, either distributed or non-distributed. In the distributed case the load balancing algorithm is run on all the nodes and the task of load balancing is shared among them. This implies that each node has to communicate with all the others, effecting the overall performance of the network.

In the non-distributed case the load balancing algorithms is done by only a single node or a group-node. Non-distributed load balancing can be run in semi-distributed form, where the nodes are grouped into clusters and each such cluster has a central node performing the load balancing. Since there is only one load balancing node the number of messages between the nodes are decreased drastically but instead we get the disadvantage of the central node becoming a single-point of failure and a bottleneck in the system. Therefore this centralized form of load balancing is only useful for small networks [6].

We will briefly mention some dynamic load balancing algorithms:

- Central Queue Algorithm - A non-distributed algorithm where the central manager maintains a cyclic FIFO-queue. Whenever a new activity arrives the manager inserts it into the queue and whenever a new request arrives it simply picks the first activity in the queue.
- Local Queue Algorithm - When a new task is created on the main host it will be allocated on under-loaded nodes. Afterwards all new tasks are allocated locally since ... [6]. The algorithm is receiver initiated since when a node is under-loaded it randomly sends requests to remote load managers
- Ant Colony Optimization Algorithm - As the name implies this algorithm is inspired by the behavior of real ants to find a optimal solution. Whenever an ant find food it moves back to the colony while leaving "markers", i.e. laying pheromone on the

way. When more ants find the same place the path the pheromone will become denser.

- Honey Bee Foraging Algorithm - This algorithm is quite similar to the Ant Colony Optimization algorithm. When bees find food they return to the bee's colony and use special dance movements for informing the other bees of how much food there is and where it's located. When the forager bees find more food a more energetic dance takes place. This phenomenon can be applied to servers, when an overloaded server receives a request it redirects it to other under loaded servers.
- Bidding - An overloaded node request bids from other nodes. The node with the best bid (i.e. lowest load) wins the job.
- Max-Min -
- Min-Min -

2.6 Task scheduling

Many studies have been recently done to improve reliability by proper task allocation in distributed systems, but they have only considered some system constraints such as processing load, memory capacity, and communication rate [11]

2.6.1 Formulas

When examine whether a certain level of reliability is reached we need to be able to put figures on the definitions of reliability. Let's consider Definition 2.1. Somehow we need to measure the probability that the hardware is functioning. One usual measurements for hardware reliability is Mean-Time Between Failure, MTBF which is defined as

$$MTBF = (nbroffailures)/(totaltime) \tag{2.2}$$

From MTBF a reliability function can be expressed as

$$R_{HW}(t) = e^{-t/MTBF} \tag{2.3}$$

which describes the probability that the hardware will work for time t without a single failure.

For measure reliability according to the dynamic definition (Definition 2.2) we will use the following formula:

$$R(t) = R_1(t) \cdot R_2(t) \cdots R_n(t) \tag{2.4}$$

where $R_k(t)$ is the probability that factor k is free from failures during time t. Some factors to consider are software (the program itself), OS (the device executing the program), hardware, network, electrical supply and load. The factors can be divided into static and dynamic factors. The static factors are those which does not change that frequently, such as electrical supply or hardware/software while the dynamic factors are those changing

more frequently, for instance the current load (or load average for last 5 minutes etc). For simplicity we will use one reliability for all static factors and chose a number of dynamic factors to consider.

In our case when we use actors we will according to Definition **??** express the reliability as:

$$R = 1 - (1 - R_{replica1}(t_0) \cdot (1 - R_{replica2}(t_0) \cdots (1 - R_{replican}(t_0) \tag{2.5}$$

where $R_{replica1}(t_0)$ is the reliability that replica 1 is reliable during the time t_0, it takes to replicate an actor. Each $R_{replica1}(t)$ can be expressed as above 2.4

2.7 Monitoring

Heartbeat-based monitors in which lack of a timely heartbeat message from the target indicates failure. Test-based monitors which send a test message to the target and wait for a reply. Messages may range from OS- level pings and SNMP queries to application level testing, e.g., a test query to a database object. End-to-end monitors which emulate actual user requests. Such monitors can identify that a problem is somewhere along the request path but not its precise location. Error logs of different types that are often produced by software components. Some error messages can be modeled as monitors which alert when the error message is produced. Statistical monitors which track auxiliary system attributes such as load, throughput, and resource utilization at various measurement points, and alarm if the values fall outside historical norms. Diagnostic tools such as filesystem and memory check- ers (e.g., fsck) that are expensive to run continuously, but can be invoked on demand. [Probabilistic Model-Driven Recovery in Distributed Systems]

Globus [9] provides a heartbeat service to monitor running processes to detect faults [15]

Chapter 3
Approach

3.1 Method

We have decided that use an action research oriented methodology. First we studied today's situation from literature and with input from our supervisor Björn Landfeldt. Therefrom we could state the background and motivation to the problem to be solved. The second step was to suggest a proper solution and implement it. Here we got input from literature and our supervisor among with Jörn och Shub. The third and most important step was to evaluate the solution in an objective way.

First of all we put together the necessary master thesis documents, a goal document and a project plan.

Thereafter we began our literature study of today's situation, as seen in 1.2 there is an certain interest in this subject...etc

During the first weeks Philip implemented a replication functionality in Calvin while Jonatan started writing the report and investigated how reliability can be defined. When the replication functionality was up and running Jonatan started writing an "actor lost" command which and deleted all the information about the lost actor used the replication functionality to start a new replica.

3.2 System model

3.2.1 Reliability model

Reliability - at least one replica is alive during the time to spawn a new replica –> 1 - P(all replicas die during the time to spawn a new replica).

Given the failure probability Pfik of each one of the mi replicas Ti k of task Ti , the new failure probability P fi0 for task Ti is: xxx The above corresponds to the probability of the

event “all the replicas and the original task fail”. Respectively, the success probability is equal to the probability of the event “the original task or at least one of its replicas executes successfully”. The number of replicas issued depends on the failure probabilities of the original tasks and on the desired fault tolerance level in the Grid infrastructure. [15]

We suppose that a mobile resource has survived until time t. The hazard function, h(t), is its failure rate during the short time interval (t , t + 1t]. Given that a mobile resource is still alive at time t , the probability of failure during the next 1t time units can be expressed generally as follows: P(t < s < t + 1t|s > t) = P(t < s < t + 1t)/P(s > t) [15]

3.2.2 Application

3.2.3 Implementation

Chapter 4
Limitations

4.0.1 Reliability model

Many engineers and researchers base their reliability models on the assumption that components of a system fail in a statistically independent manner. This assumption is often violated in practice because environmental and system specific factors contribute to correlated failures, which can lower the reliability of a fault tolerant system. A simple method to quantify the impact of correlation on system reliability is needed to encourage models explicitly incorporating correlated failures. Previous approaches to model correlation are limited to systems consisting of two or three components or assume that the majority of the subsets of component failures are statistically independent. [14]

the interrelationship constraint between task modules has not been taken into account in calculation of distributed system reliability (DSR). In distributed simulation, the LPs simulation advances are bound by synchronization constraint, which will affect the executive time of system components. [10]

We do not take into account parameters such as X and Y [TODO].

Furthermore failure data which they have collected for their experiments through in-house testing cannot be compared with failures that can occur under actual operational environment [2]

A lot of researchers are of the view that service providers must provide some other detail to compute reliability of both type of services i.e. atomic service and composite service. Such as, external services it uses, how service are glued together in composite service, how frequently they call each other, and flow graph describing behavior of service [2]. Service Oriented Reliability Model (SORM) [18] computed reliability of atomic and composite services exploiting distinct technique [2]

Given the failure probability Pfik of each one of the mi replicas Ti k of task Ti , the new failure probability P fi0 for task Ti is: 0 mi Y Pfi =Pfi· Pfik. (2) k=1 The above corresponds to the probability of the event “all the replicas and the original task fail”. Respectively, the success probability is equal to the probability of the event “the original task or at least

one of its replicas executes successfully". The number of replicas issued depends on the failure probabilities of the original tasks and on the desired fault tolerance level in the Grid infrastructure. [15]

Conventionally, MTTF refers to non-repairable objects, while mean time between failure (MTBF) refers to repairable objects. In large and complex systems such as Mobile Grids, although the individual component reliabilities can be high, the overall reliability of the system can possibly be low [34], due to coupled failure modes or various extrinsic factors. The lifetime of the Mobile Grid resources under discussion is assumed to start at time $t = 0$, and any events occurring at times $t < 0$ are irrelevant. [15]

Chapter 5
Future Work

5.0.1 Extended model

Reliability of server's availability and scala- bility (such as File Server, DB servers, Web servers, and email servers, etc.), communica- tion infrastructure, and connecting devices. [2]

Chapter 6
Evaluation

Chapter 7
Conclusions

Bibliography

[1] Sol M. Shatz, Jia-Ping Wang and Masanori Goto, *Task Allocation for Maximizing Reliability of Distributed Computer Systems*, Computers, IEEE Transactions on, Volume 41 Issue 9, Sep 1992

[2] Waseem Ahmed and Yong Wei Wu, *A survey on reliability in distributed systems*, Journal of Computer and System Sciences Volume 78 Issue 8, December 2013, Pages 1243–1255

[3] Mayanka Katyal and Atul Mishra, *A Comparative Study of Load Balancing Algorithms in Cloud Computing Environmen*, International Journal of Distributed and Cloud Computing, Volume 1 Issue 2, 2013

[4] Michael Treaster, *A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems*, Cornell University Library, Jan 2005

[5] Soonwook Hwang and Carl Kesselman, *Grid Workflow: A Flexible Failure Handling Framework for the Grid*, High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on, p. 126-137, June 2003

[6] Prashant D. Maheta , Kunjal Garala and Namrata Goswami, *A Performance Analysis of Load Balancing Algorithms in Cloud Environment*, Computer Communication and Informatics (ICCCI), 2015 International Conference on, Jan 2015

[7] Shuli Wang et. al. *A Task Scheduling Algorithm Based on Replication for Maximizing Reliability on Heterogeneous Computing Systems*, Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, p. 1562 - 1571, May 2014

[8] Distributed workflow mapping algorithm for maximized reliability under end-to-end delay constraint

[9] Matching and Scheduling Algorithms for Minimising Execution Time and Failure Probability of Applications in Heterogeneous Computing

[10] Reliability model of distributed simulation system

[11] Optimal task allocation for maximizing reliability in distributed real-time systems

[12] Performance Implications of Periodic Checkpointing on Large-scale Cluster Systems

[13] A large-scale study of failures in high-performance computing systems

[14] Discrete and continuous reliability models for systems with identically distributed correlated components

[15] Efficient task replication and management for adaptive fault tolerance in Mobile Grid environments

[16] Real-time fault-tolerant scheduling algorithm for distributed computing systems

Appendices

Appendix A

A

Chapter 8
Formatting

Avoid empty spaces between *chapter-section*, *section-sub-section*. For instance, a very brief summary of the chapter would be one way of bridging the chapter heading and the first section of that chapter.

8.1 Page Size and Margins

Use A4 paper, with the text margins given in Table 8.1.

Table 8.1: Text margins for A4.

margin	space
top	3.0cm
bottom	3.0cm
left (inside)	2.5cm
right (outside)	2.5cm
binding offset	1.0cm

8.2 Typeface and Font Sizes

The fonts to use for the reports are **TeX Gyre Termes** (a **Times New Roman** clone) for serif fonts, **TeX Gyre Heros** (a **Helvetica** clone) for sans-serif fonts, and finally **`TeX Gyre Cursor`** (a **`Courier`** clone) as mono-space font. All these fonts are included with the TeXLive 2013 installation. Table 8.2 lists the most important text elements and the associated fonts.

Table 8.2: Font types, faces and sizes to be used.

Element	Face	Size	LaTeXsize
Ch. label	**serif, bold**	24.88pt	`\huge`
Chapter	**serif, bold**	24.88pt	`\Huge`
Section	**sans-serif, bold**	20.74pt	`\LARGE`
Subsection	**sans-serif, bold**	17.28pt	`\Large`
Subsubsection	**sans-serif, bold**	14.4pt	`\large`
Body	serif	12pt	`\normalsize`
Header	serif, SmallCaps	10pt	
Footer (page numbers)	serif, regular	12pt	
Figure label	**serif, bold**	12pt	
Figure caption	serif, regular	12pt	
In figure	sans-serif	*any*	
Table label	**serif, bold**	12pt	
Table caption and text	serif, regular	12pt	
`Listings`	`mono-space`	$\leq$ 12pt	

8.2.1 Headers and Footers

Note that the page headers are aligned towards the outside of the page (right on the right-hand page, left on the left-hand page) and they contain the section title on the right and the chapter title on the left respectively, in SmallCaps. The footers contain only page numbers on the exterior of the page, aligned right or left depending on the page. The lines used to delimit the headers and footers from the rest of the page are 0.4*pt* thick, and are as long as the text.

8.2.2 Chapters, Sections, Paragraphs

Chapter, section, subsection, etc. names are all left aligned, and numbered as in this document.

Chapters always start on the right-hand page, with the label and title separated from the rest of the text by a 0.4*pt* thick line.

Paragraphs are justified (left and right), using single line spacing. Note that the first paragraph of a chapter, section, etc. is not indented, while the following are indented.

8.2.3 Tables

Table captions should be located above the table, justified, and spaced 2.0cm from left and right (important for very long captions). Tables should be numbered, but the numbering is up to you, and could be, for instance:

- **Table X.Y** where X is the chapter number and Y is the table number within that chapter. (This is the default in LaTeX. More on LaTeX can be found on-line, including whole books, such as [**?**].) or

- **Table Y** where Y is the table number within the whole report

As a recommendation, use regular paragraph text in the tables, bold headings and avoid vertical lines (see Table 8.2).

8.2.4 Figures

Figure labels, numbering, and captions should be formed similarly to tables. As a recommendation, use vector graphics in figures (Figure 8.1), rather than bitmaps (Figure 8.2). Text within figures usually looks better with sans-serif fonts.

Figure 8.1: A PDF vector graphics figure. Notice the numbering and placement of the caption. The caption text is indented 2.0cm from both left and right text margin.

Figure 8.2: A JPEG bitmap figure. Notice the bad quality of such an image when scaling it. Sometimes bitmap images are unavoidable, such as for screen dumps.

For those interested in delving deeper into the design of graphical information display, please refer to books such as [**?**, **?**].

8.3 Mathematical Formulae and Equations

You are free to use in-text equations and formulae, usually in *italic serif* font. For instance: $S = \sum_i a_i$. We recommend using numbered equations when you do need to refer to the specific equations:

$$E = \int_0^{\delta} P(t)dt \quad \longleftrightarrow \quad E = mc^2 \tag{8.1}$$

The numbering system for equations should be similar to that used for tables and figures.

8.4 References

Your references should be gathered in a **References** section, located at the end of the document (before **Appendices**). We recommend using number style references, ordered as appearing in the document or alphabetically. Have a look at the references in this template in order to figure out the style, fonts and fields. Web references are acceptable (with restraint) as long as you specify the date you accessed the given link [**?**, **?**]. You may of course use URLs directly in the document, using mono-space font, i.e. `http://cs.lth.se/`.

8.5 Colours

As a general rule, all theses are printed in black-and-white, with the exception of selected parts in selected theses that need to display colour images essential to describing the thesis outcome (*computer graphics*, for instance).

A strong requirement is for using **black text on white background** in your document's main text. Otherwise we do encourage using colours in your figures, or other elements (i.e. the colour marking internal and external references) that would make the document more readable on screen. You may also emphasize table rows, columns, cells, or headers using white text on black background, or black text on light grey background.

Finally, note that the document should look good in black-and-white print. Colours are often rendered using monochrome textures in print, which makes them look different from on screen versions. This means that you should choose your colours wisely, and even opt for black-and-white textures when the distinction between colours is hard to make in print. The best way to check how your document looks, is to print out a copy yourself.

Chapter 9
Language

You are strongly encouraged to write your report in English, for two reasons. First, it will improve your use of English language. Second, it will increase visibility for you, the author, as well as for the Department of Computer Science, and for your host company (if any).

However, note that your examiner (and supervisors) are not there to provide you with extensive language feedback. We recommend that you check the language used in your report in several ways:

Reference books dedicated to language issues can be very useful. [**?**]

Spelling and grammar checkers which are usually available in the commonly used text editing environments.

Colleagues and friends willing to provide feedback your writing.

Studieverkstaden is a university level workshop, that can help you with language related problems (see Studieverkstaden's web page).

Websites useful for detecting language errors or strange expressions, such as

- `http://translate.google.com`
- `http://www.gingersoftware.com/grammarcheck/`

9.1 Style Elements

Next, we will just give some rough guidelines for good style in a report written in English. Your supervisor and examiner as well as the aforementioned **Studieverkstad** might have a different take on these, so we recommend you follow their advice whenever in doubt. If you want a reference to a short style guide, have a look at [**?**].

Widows and Orphans

Avoid *widows* and *orphans*, namely words or short lines at the beginning or end of a paragraph, which are left dangling at the top or bottom of a column, separated from the rest of the paragraph.

Footnotes

We strongly recommend you avoid footnotes. To quote from [**?**], *Footnotes are frequently misused by containing information which should either be placed in the text or excluded altogether. They should be avoided as a general rule and are acceptable only in exceptional cases when incorporation of their content in the text [is] not possible.*

Active vs. Passive Voice

Generally active voice (*I ate this apple.*) is easier to understand than passive voice (*This apple has been eaten (by me).*) In passive voice sentences the actor carrying out the action is often forgotten, which makes the reader wonder who actually performed the action. In a report is important to be clear about who carried out the work. Therefore we recommend to use active voice, and preferably the plural form *we* instead of *I* (even in single author reports).

Long and Short Sentences

A nice brief list of sentence problems and solutions is given in [**?**]. Using choppy sentences (too short) is a common problem of many students. The opposite, using too long sentences, occurs less often, in our experience.

Subject-Predicate Agreement

A common problem of native Swedish speakers is getting the subject-predicate (verb) agreement right in sentences. Note that a verb must agree in person and number with its subject. As a rough tip, if you have subject ending in *s* (plural), the predicate should not, and the other way around. Hence, *only one s*. Examples follow:

incorrect He have to take this road.

correct He has to take this road.

incorrect These words forms a sentence.

correct These words form a sentence.

In more complex sentences, getting the agreement right is trickier. A brief guide is given in the *20 Rules of Subject Verb Agreement* [**?**].

Chapter 10
Structure

It is a good idea to discuss the structure of the report with your supervisor rather early in your writing. Given next is a generic structure that is a starting point, but by no means the absolute standard. Your supervisor should provide a better structure for the specific field you are writing your thesis in. Note also that the naming of the chapters is not compulsory, but may be a helpful guideline.

Introduction should give the background of your work. Important parts to cover:

- Give the context of your work, have a short introduction to the area.
- Define the problem you are solving (or trying to solve).
- Specify your contributions. What does this particular work/report bring to the research are or to the body of knowledge? How is the work divided between the co-authors? (This part is essential to pinpoint individual work. For theses with two authors, it is compulsory to identify which author has contributed with which part, both with respect to the work and the report.)
- Describe related work (literature study). Besides listing other work in the area, mention how is it related or relevant to your work. The tradition in some research area is to place this part at the end of the report (check with your supervisor).

Approach should contain a description of your solution(s), with all the theoretical background needed. On occasion this is replaced by a subset or all of the following:

- **Method**: describe how you go about solving the problem you defined. Also how do you show/prove that your solution actually works, and how well does it work.
- **Theory**: should contain the theoretical background needed to understand your work, if necessary.

- **Implementation**: if your work involved building an artefact/implementation, give the details here. Note, that this should not, as a rule, be a chronological description of your efforts, but a view of the result. There is a place for insights and lamentation later on in the report, in the Discussion section.

Evaluation is the part where you present the finds. Depending on the area this part contains a subset or all of the following:

- **Experimental Setup** should describe the details of the method used to evaluate your solution(s)/approach. Sometimes this is already addressed in the **Method**, sometimes this part replaces **Method**.
- **Results** contains the data (as tables, graphs) obtained via experiments (benchmarking, polls, interviews).
- **Discussion** allows for a longer discussion and interpretation of the results from the evaluation, including extrapolations and/or expected impact. This might also be a good place to describe your positive and negative experiences related to the work you carried out.

Occasionally these sections are intermingled, if this allows for a better presentation of your work. However, try to distinguish between measurements or hard data (results) and extrapolations, interpretations, or speculations (discussion).

Conclusions should summarize your findings and possible improvements or recommendations.

Bibliography is a must in a scientific report. LaTeX and `bibtex` offer great support for handling references and automatically generating bibliographies.

Appendices should contain lengthy details of the experimental setup, mathematical proofs, code download information, and shorter code snippets. Avoid longer code listings. Source code should rather be made available for download on a website or on-line repository of your choosing.

Appendices

Appendix B
About This Document

The following environments and tools were used to create this document:

- operating system: Mac OS X 10.10.1
- tex distribution: MacTeX-2014, `http://www.tug.org/mactex/`
- tex editor: Texmaker 4.4.1 for Mac, `http://www.xm1math.net/texmaker/` for its XeLaTeX flow (recommended) or pdfLaTeX flow
- bibtex editor: BibDesk 1.6.3 for Mac, `http://bibdesk.sourceforge.net/`
- fonts `cslthse-msc.cls` document class):

  for XeLaTeX: TeX Gyre Termes, TeX Gyre Heros, `TeX Gyre Cursor` (installed from the TeXLive 2013)

  for pdfLaTeX: TeX Gyre font packages: tgtermes.sty, tgheros.sty, tgcursor.sty, gtxmath.sty (available through TeXLive 2013)
- picture editor: OmniGraffle Professional 5.4.2

A list of the essential LaTeXpackages needed to compile this document follows (all except `hyperref` are included in the document class):

- `fontspec`, to access local fonts, needs the XeLaTeX flow
- `geometry`, for page layout
- `titling`, for formatting the title page
- `fancyhdr`, for custom headers and footers
- `abstract`, for customizing the abstract

- `titlesec`, for custom chapters, sections, etc.
- `caption`, for custom tables and figure captions
- `hyperref`, for producing PDF with hyperlinks
- `appendix`, for appendices
- `printlen`, for printing text sizes
- `textcomp`, for text companion fonts (e.g. bullet)

Other useful packages:

- `listings`, for producing code listings with syntax colouring and line numbers

Appendix C
List of Changes

Since 2015/04/27

- Improved the **Structure** chapter and added more detailed comments for each part.

Since 2014/02/18

- Added the possibility to specify two supervisors. Use either of the `\supervisor{}` or `\supervisors{}{}` commands to set the names and contacts on the first page.

Since 2013/09/23

- Added missing colon ":" after *Examiner* on the front page.

Since 2013/08/30

- Changed fonts from Garamond (Times New Roman), Helvetica (Arial), Courier (Source Code Pro) to Tex Gyre fonts, namely Termes, Heros, Cursor, which are freely available with TexLive 2013 installation. These are all clones of Times New Roman, Helvetica and Courier, respectively. Garamond is problematic on some systems, being a non-freely available font.

- Corrected the *Face* column in Table 8.2 to correctly depict the font face.

Since 2013/02/22

- Number of words required in the abstract changed to 150 (from 300).

Since 2013/02/15

- Made a separate document class, for clarity.
- made it work with pdfLaTeX and garamond.sty, in addition to XeLaTeX and true type fonts. It is up to the user to get the hold of the garamond.zip from `http://gael-varoquaux.info/computers/garamond/index.html`.