# Scalable applications in a distributed environment

Simon Norberg - norberg.simon@gmail.com
Filip Andersson - flpandersson@gmail.com

Blekinge Institute of Technology

May 26, 2011

*It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change.*

*- -Charles Darwin (1809-1882)*

**Abstract**

As the amount of simultaneous users of distributed systems increase, scalability is becoming an important factor to consider during software development. Without sufficient scalability, systems might have a hard time to manage high loads, and might not be able to support a high amount of users. We have determined how scalability can best be implemented, and what extra costs this leads to. Our research is based on both a literature review, where we have looked at what others in the field of computer engineering thinks about scalability, and by implementing a highly scalable system of our own. In the end we came up with a couple of general pointers which can help developers to determine if they should focus on scalable development, and what they should consider if they choose to do so.

# 1 Introduction

Distribution and scalability is becoming an important issue in todays application development [1]. The need for an application to scale between a few users up to several millions on very short notice gives developers limited time to modify and/or redesign the application, so having an already scalable system has a lot of benefits. If a system looses its ability to scale, and thereby cannot support more users, it may quickly loose its target audience to a competitor. Furthermore, being able to modify and update parts of the system separately, rather than everything at once, makes application maintenance significantly easier and downtime minimal in production systems. The focus of this thesis is to determine how one can successfully develop a scalable system, and what extra cost and effort this requires compared to developing a non-scalable system. We feel that this is important since not all systems benefit from being scalable, and thus the eventual extra costs may not be justified. On the other hand, if a system would benefit from being scalable, but is not, developers might risk loosing a lot in the long run. We are looking to solve this problem by proposing a few pointers on things we find to be important to consider, both when trying to determine however one should develop scalable or not, and if so, how it can be done. We will be researching this by doing a literature review, where we analyze what others has to say on the subject, and a post-mortem analysis, where we try to estimate the benefits and costs of scalability by developing a scalable system of our own.

## 1.1 Overview

The ”Background” section outlines the concept of scalability, as well as some related background concepts.

The ”Research Questions” section defines our research questions, i.e. the questions we aim to answer in this thesis.

The ”Research Methodology” section outlines and explains the methodologies we have used to conduct our research, which are literature review and post mortem analysis.

The "Post mortem results" section presents the data gathered from our post mortem analysis.

The "Literature review results" section presents the design of our literature review, as well as the articles found and selected during our literature review.

The "Literature Review Findings" section presents our findings from the literature review.

In the "Analysis and Discussion" section we discuss our findings, both from the literature review and from the post mortem analysis.

In the "Conclusion" section we present the conclusions of our research.

Our selected papers can be found in appendix A.

## 1.2 Background

In this section we explain the concept of scalability, as well as some general background concepts related to the subject.

A scalable system is a system that is flexible and adaptive to changing conditions, such as an increase in system load, new system components etc., and that can be economically deployed in many configurations, small as well as large [2]. The scalability of a system depends on how well it can exploit functional modularity, structural regularity and hierarchy [3].

*Distributed Computing*, perhaps more known as *Cloud Computing*, is becoming a more and more common approach to software development. The basic concept of distributed computing is to make different services available in the so called 'cloud', which means that these services are reachable from anywhere one has internet access. These services can, for example, be things like e-mail, document editing, social networks and much more. The fact that these services are so widely distributed places high demands on the number of users that can be supported. These demands can be accommodated by simply making sure the systems have a large amount of resources from the start, or adding resources from time to time. However, if there's less users on the system than it has capacity for, this approach quickly becomes expensive due to a lot of unused resources. This problem can be solved by making the system scalable, i.e. that it uses resources in proportion to the number of current users. If the user base drops, it will use less resources, and when more people are using the system it will use more. By making a system scalable, one makes it able to adapt to changing conditions and environments, which is important since a system is generally developed to run over a relatively large period of time, during which a lot of things may rapidly change [4] [5]. This can be crucial to, among other things, reduce software development cost [6].

To promote scalability in a distributed setting one can use, for example, the *Message Passing Interface (MPI)* which enables message passing between parallel programs running on computer clusters. MPI is an well tested and de facto[7] standard for writing scalable software in any language and on any platform. MPI makes it easy to delegate work in an scalable and distributed environment and at the same time collect the results form different nodes. MPI

also allows an arbitrary number of nodes in the system which make it easy to add more resources when performance needs to be improved. [8]
MPI works by distributing part of the computations to different instances that could be running on the same computer and on different cores or even completely different computer architectures connected over the Internet. MPI provides facilities to sending the results from this computations to other instances within MPI to be used as input for other computations or to be presented to the user. There exist several alternatives[9] to MPI, some of them being ARMIC, Sandia Portals and GASNet. Yet MPI is the de facto [9] standard for writing massively parallel applications. If you only want to write parallel applications to take advantage of the local computer, running several ordinary threads communicating over an application specific method (often shared memory) is still very common [10], even if libraries like MPI exist to make development much easier.

We are going to use MPI in python utilizing the boost projects MPI bindings for python. These bindings allow us to use MPI in python even if the MPI library is written in another programming language, in this case C++.

## 1.3 Goals

What we intend to investigate is what needs to be considered when developing a scalable distributed system, and to what cost these features comes at. We believe that while the cost of these features may initially be higher, the cost of maintenance is lower. We hope that the outcome of our research will be useful for people trying to determine whether or not to strive towards implementing a scalable distributed system from the start, rather than incrementally increasing system capacity during the system's lifetime.

# 2 Research questions

- RQ1: How do one successfully develop a scalable distributed system? Which factors needs to be considered, and how does one achieve these? Without knowing what needs to be done, developing a high-grade scalable system may be difficult, and cost a lot of extra time.
- RQ1.1: Which technologies, libraries and software can one use to achieve high-grade scalability? Not all technologies etc. support a high level of scalability, and knowing the difference between the ones who do and the ones who do not can save a lot of expenses.
- RQ2: Which extra costs do the development of scalable systems lead to? How much does it cost when doing it in advance, and how much does it cost when doing it after the initial development?
- RQ3: Which kind of systems can benefit from having high-grade scalability? It is important to know if a certain system actually benefits from being highly scalable, and if so, how much. If the benefits are relatively small, there might be cheaper ways of developing it.

# 3 Research methodology

We will be using a Literature Review and a Post-mortem analysis to conduct our research.

## 3.1 Post-mortem analysis design

### 3.1.1 MPI

We will study a classical and well tested approach to writing scalable distributed systems with MPI [8]. MPI have easy to use python bindings [11] [12], which we will be using to write test programs that test the capabilities of MPI, and compare the performance and cost to develop compared to a ordinary non scalable application.

**Problem description** The problem we going to try to solve is from a hash of a user password try to bruteforce (testing all possible combinations of letters and numbers to match a password) and find out what password was used to generate the hash. Since a hash is a one-way algorithm (or at least should be), we have to test every possible password and check if the hash matches the hashed user password, and if it matches we have probably[1] found the same password as the user had.

[1] We could possibly have found a collision in the algorithm, but the chance is minimal in a well tested algorithm like sha256 and in the relative small password range we're testing, only $\sim$ 40 million

In the application without MPI we're going to do it by iterating over all words in a dictionary, hashing them one by one, while attempting to match it to the hash we are trying to bruteforce.

With MPI we will create several instances of MPI that each iterates over a part of the dictionary simultaneously.

We will be measuring:

- Time to develop
  - **Why** - If we know how much time it takes to develop a scalable application with MPI we can measure the cost of developing the scalable application. Which is needed to know in which cases it may be cost efficient.
  - **How** - By keeping track of spent time for writing the software.
- Lines of code
  - **Why** - The easiest metric of how complex and hard to write and maintain a program is. The metric is a bit problematic if you compare code written by different programmers, some write short and hard to read code and other more verbose but easier to read. But by letting the same programmers write both of the applications we hope to overcome that difference.
  - **How** - Count number of lines of code.
- How easy the code is to understand
  - **Why** - Code that is easy to understand is also easy to maintain and extend.
  - **How** - Hard to objectively measure so were going to measure the Cyclomatic Complexity [13]
- How easy it is to extend the program
  - **Why** - If the application becomes harder to extend and improve upon when using MPI it maybe isn't feasible to write applications that needs to be extended further on with MPI.
  - **How** - Measure how much work it takes to extend the application with some example feature
- Performance - How fast does the application run
  - **Why** - Because one of the main reason for scalability is to increase the performance we want to know how much we are able to increase it by using MPI.
  - **How** - Measure the time to run the program with *time* (a UNIX command)

- Scalability - how well do it scale when adding more resources
  - **Why** - Optimal we would like to have an application that have perfect scalability [2] so when doubling the resources the performance also doubles. This is probably not the case, but we want to know how much performance gain we get by adding more resources to the system.
  - **How** - Adding more resources to the system and then measuring performance again
- Performance when using only one node, e.g. overhead for MPI
  - **Why** - We want to know how much performance penalty we get for using MPI in a system that do not need the scalability of MPI, either because it is less usage at the time (e.g. due to it being night) or because the system don't have a high usage yet and maybe never will.
  - **How** - Setting up MPI with just one node and measure the performance compared to the non scalable application.

# 4 Post mortem results

## 4.1 Pre study

When we started by look at the different MPI bindings provided for python, the results were overwhelming and we found several different bindings providing the same functionality. After some research we chose to use boost-mpi's python bindings [14]. These bindings looked well maintained, had a proper debian package, and had example code that worked out of the box, as opposed to the other bindings we tested. The problem was that it existed more than one library with the exact same name but was used differently and it was hard to know what library different examples was supposed to work with.

## 4.2 Problem

Before we could write a scalable application we had to choose what problem we wanted to solve with our application; we chose to write an application that would try to find what password a user had from the hash it generated. The reason is that it requires a lot of cpu time to calculate and it is very easy to split up in several parts and doing them parallel.

## 4.3 Solution

The solutions works by iterating over a dictionary of ∼3.6 Million words and test if the word matches the supplied hash and also check if a word is followed by a digit match. The reason that we also check the word followed by a digit is to increase the load on the system due to the execution time being to low to reliable measure and reproduce the data when only hashing ~3.6 Million words.

## 4.4 Set up

2 Computers with:

- Intel(R) Core 2 Duo CPU6600 @ 2.40GHz
- 2 GB Memory
- Local 1 Gbit lan
- Debian Squeeze
- mpich 1.2.7-9.1
- boost-mpi-python 1.42.0
- Our test application [15]

## 4.5 Measuring

Measuring was done by running the application 10 times for every number of instances. We then calculated the mean time for each number of instances. A non parallel program was also written and used to compare to our parallel application.

### 4.5.1 Scalability

This table and diagram shows how much performance is increased for every new instance of MPI stared, compared to how well it should have performed if it had Perfect scalability[2]. Perfect Scalability is when you increase the resources, the performance is increased just as much.

| Instances | python MPI (sec) | Perfect Scalability (sec) |
|---|---|---|
| 1 | 101,48 | 101,48 |
| 2 | 53,66 | 50,74 |
| 3 | 36,03 | 33,83 |
| 4 | 28,37 | 25,37 |
| 5 | 34,24 | 25,37 |
| 6 | 29,32 | 25,37 |
| 7 | 31,57 | 25,37 |
| 8 | 29,67 | 25,37 |
| 9 | 31,73 | 25,37 |
| 10 | 29,94 | 25,37 |

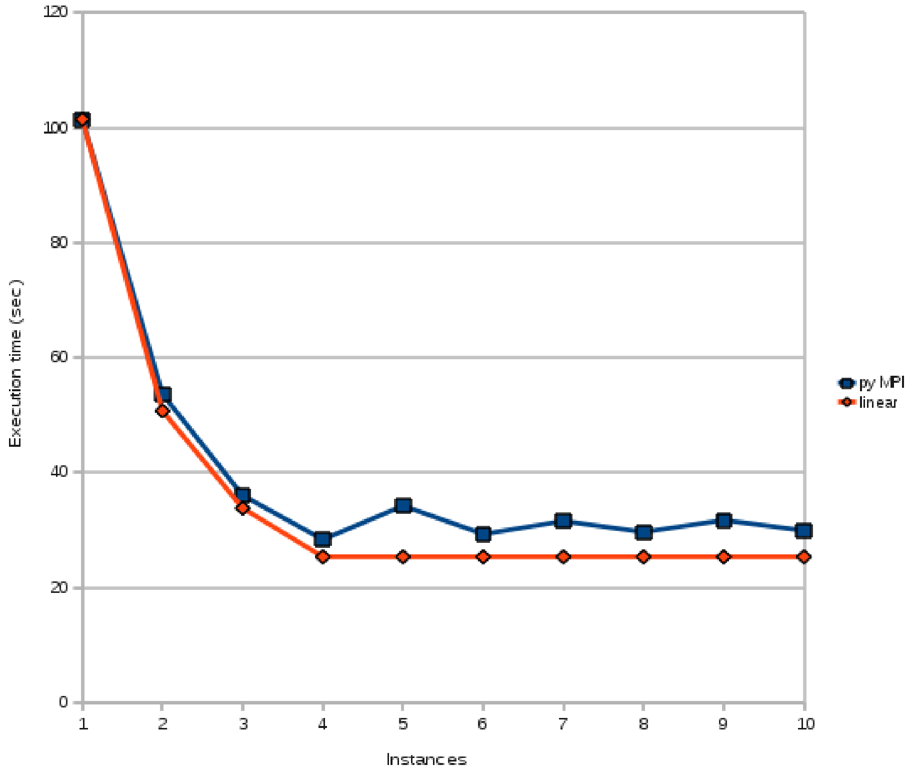

Figure 1: Total number of cores in the system was 4, so performance stopped increasing when adding more instances than 4.

We can see that the performance almost increase linear when adding more resources to the system. This shows that our application don't have any major bottlenecks, and it would be easy to increase the performance of the application by just adding more resources when needed.

### 4.5.2 Scalable vs. Non Scalable Systems

This table and diagrams compare our scalable application towards the non scalable application we wrote for comparison.

| Instances | python MPI (sec) | Non Scalable System (sec) |
|---|---|---|
| 1 | 101,48 | 96.82 |
| 2 | 53,66 | 96.82 |
| 3 | 36,03 | 96.82 |
| 4 | 28,37 | 96.82 |
| 5 | 34,24 | 96.82 |
| 6 | 29,32 | 96.82 |
| 7 | 31,57 | 96.82 |
| 8 | 29,67 | 96.82 |
| 9 | 31,73 | 96.82 |
| 10 | 29,94 | 96.82 |

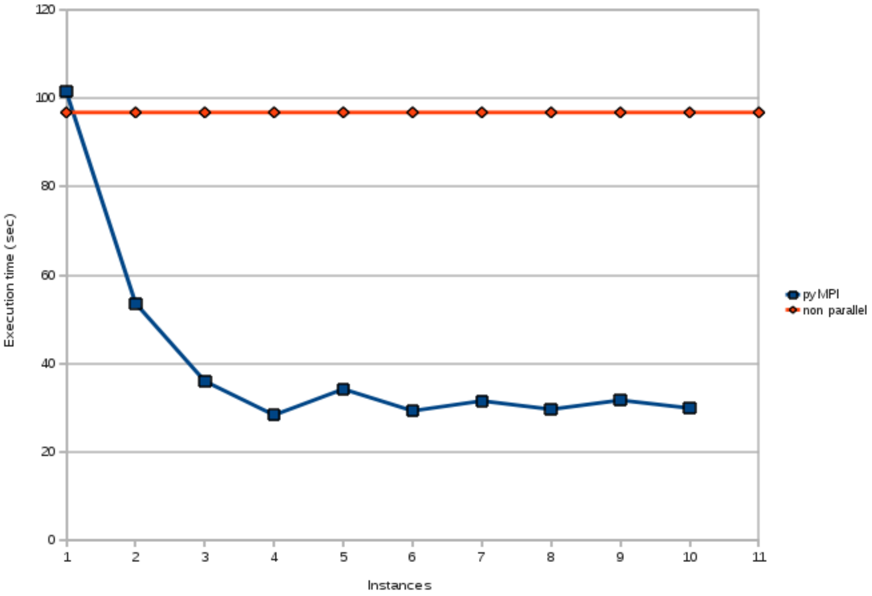

Figure 2: From this we can see that we have a small performance loss (5%) due the usage of MPI but if we have more than one core to run on we get drastically better performance.

#### 4.5.3 Complexity

We have measured the complexity of both our parallel and non parallel application, and presents the results below. One of the measurements we do is McCabe Complexity[13] or better know as Cyclomatic complexity. It tries to measure the complexity of the application by measuring the number of independent paths through the code that could be taken in a program. A program with a lot of different possible paths is likely to be more complex than one with very few.

The metric is however not perfect since it don't reflect anything else, e.g. a program written on one row with cryptic variable names may have the same Cyclomatic complexity as a program structured in a more human readable way over several lines with descriptive variable names.

| Measurement | python MPI | Non Scalable System |
|---|---|---|
| Lines of code | 23 | 21 |
| Block Depth | 5 | 5 |
| Blocks | 9 | 9 |
| Tokens | 267 | 242 |
| McCabe Complexity __main__ | 2 | 2 |
| McCabe Complexity find_password | 7 | 7 |

Table 1: Measurements done with pyMetrics[16]

From these measurements we can see that the differences is minimal and McCabe Complexity[13] don't differ at all between the applications. We also made a small survey in our project of 6 persons and found that no one thought that differed much in readability, the only thing was that it was 2 lines more code in the parallel version which they therefore saw it as a little bit more difficult.

Below we show how the code looks to allow the reader to compare the readability of the different applications.

### Common part of the application

```
import random, hashlib
from glob import glob

def find_password(brute_hash, start_chunk, nr_of_instances):
        files = glob("pympi-test/dicts/x*")
        for i in range(start_chunk, len(files), nr_of_instances):
                words = open(files[i])
                for word in words.readlines():
                        for i in range(-1,10):
                                if i == -1:
                                        hash_word = word.strip()
                                else:
                                        hash_word = word.strip() + str(i)
                                new_hash = hashlib.sha256(hash_word).hexdigest()
                                if new_hash == brute_hash:
                                        return hash_word
        return None
```

#### Parallel version of the application

```
from find_password import find_password
import hashlib
from boost import mpi

if __name__=="__main__":
        rank, size = mpi.rank, mpi.size
        passwd = find_password(hashlib.sha256("simon22").hexdigest(), rank, size)
        print passwd
```

#### Non Parallel version of the application

```
import hashlib
from find_password import find_password

if __name__=="__main__":
        passwd = find_password(hashlib.sha256("simon22").hexdigest(), 0, 1)
        print passwd
```

### 4.5.4 Time

Below we present how much time was spent on developing the different applications in a 5 minute resolution. "Time to develop" was the time spent on the initial application that hashed 3.6 million words from a word list. "Time to extend" is the time it took to extend the application to also test every word with a digit at the end.

| Measurement | python MPI (min) | Non Scalable System(min) |
|---|---|---|
| Time to develop | 60 | 60 |
| Time to extend | 15 | 15 |

Table 2: These times don't include time spent on prestudy, but rather the time it takes for someone with at least basic understanding of python and MPI. The pre study for MPI took about 1-2 hours for someone without previous experience to MPI.

We can see that using MPI in this type of applications does not require any extra effort from the programmer, if he or she has some previous knowledge about MPI. But in applications that isn't as easy to parallelize the difference may be significant to MPI's disadvantage.

# 5 Literature review results

## 5.1 Literature review design

We will be searching for literature for our review mainly on Google Scholar and on BTH's article database Elin. The keywords we will be using while searching will be focused on terms related to distributed systems and scalability.

The research started with the keywords scalable and distributed systems, from there we read article that looked promising first by just looking on the topic and if it looked relevant for our thesis, we took a closer look at the abstract. After reading the abstract we made a decision if the paper was in the scope of our project and if it was that we selected it. And used terminology found in that paper to find further papers on which the same process was used.

List of keywords:

- scalable systems (Google Scholar)
- python parallel (Google Scholar)
- mpi (Google Scholar)
- Distributed systems AND scalable design (ELIN)
- Distributed environments services osgi (ELIN)
- Scalable distribution AND software (ELIN)
- Scalable development AND cost (ELIN)
- Scalability AND distributed systems (ELIN)
- Distributed Applications AND Software Modularization (ELIN)

# 6 Literature Review Findings

## 6.1 Summary of findings

None of the findings are, as we see it, contradictive to one another, i.e. we have found no opposing results. This indicates that there is somewhat of a consensus regarding as to what scalability is, what the advantages and disadvantages of it are, and how it is best achieved. As we see it, this is a good thing, since it gives us a clear picture of how to reach the goals of this paper, and saves us the trouble of having to match the opinions of different authors against one another to try to determine who's right and who's wrong. We feel confident that all our findings are valid and relevant to our research, and that we, with the help of these findings, will be able to answer our research questions and thus achieve our goal with this thesis. We also located a number of findings (in [8][11][12]) that serves well as a reference in our post-mortem analysis, since they talk about scalability that is achieved using the same technology as we are using.

## 6.2 Mapping between articles and Research Questions

This is a table which maps the articles we've found with the Research Question(s) (RQ) they answer. For example, the article ”Evaluating the scalability of distributed systems” answers Research Question 3.

| Paper | Research Questions |
|---|---|
| Evaluating the scalability of distributed systems | RQ3 |
| Principles of modularity, regularity and hierarchy for scalable systems | RQ1, RQ3 |
| Software engineering for scalable distributed applications | RQ1, RQ2 |
| Towards a scalable design for command and control systems | RQ1, RQ3 |
| Towards Scalable and Adaptable Software Architectures | RQ1, RQ2 |
| A high-performance, portable implementation of the MPI message passing interface standard | RQ1, RQ1.1, PM-MPI |
| pyMPI - An introduction to parallel Python using MPI | RQ1, RQ1.1, PM-MPI |
| Parallel, distributed scripting with python | RQ1, RQ1.1, PM-MPI |
| Validity of the single processor approach to achieving large scale computing capabilities | RQ3 |
| Ten Performance Guidelines for Balancing Software Quality Attributes when Developing Large Real-time Applications for Multiprocessors | RQ1, RQ2, RQ3 |

Table 3: RQn = which research question the paper answers, PM-MPI = papers related to Post Mortem

## 6.3 Findings

### Research Question 1 and 2

*Research Question 1: How do one successfully develop a scalable distributed system? Which factors needs to be considered, and how does one achieve these? Without knowing what needs to be done, developing a high-grade scalable system may be difficult, and cost a lot of extra time.*

*Research Question 2: Which extra costs do the development of scalable systems lead to? How much does it cost when doing it in advance, and how much does it cost when doing it after the initial development?*

van Steen, van der Zijden and Sips[4] defines a scalable system as a system that ”should easily be able to accommodate higher performance levels”, and argues that ”systems should be scalable in the sense that they can easily be adapted to cooperate with future applications”. On the subject of the cost of scalability, they state that one important factor in scalable development ”are the additional bounds that limit the increase of costs, and the degradation of performance, respectively”, i.e. that if there are heavy constraints on development costs, scalability (in the form of performance) will suffer. They also argue that one of the major problems with scalable development is that the solution as to how it is done depends heavily on the application itself. They feel that constructing ”general-purpose scalable solutions for replicating and distributing data and functionality, does not make much sense”.

Fayad, Hamza and Sanchez [6] argues that there are several types of architectures one can use while developing a scalable system, most notably the ones they refer to as *"Upward and Downward Scalability"*. They define upward scalability as "the capacity of handling increasing demands by adding new layers or functionalities" and downward scalability as "the ability of the architecture to adapt itself to a more constrained environment, by disabling certain functionalities". They define three core elements one should consider when trying to achieve upward scalability as being *Increased capacity and speed*, *Improved efficiency* and *Reduced cost*.

They also suggest a number of techniques that can be used in order to address these elements:

- "To achieve *Increased capacity and speed* one can use faster machines, create a machine cluster, and use appliance servers."
- "To *Improve efficiency* one can use appliance servers, segment the workload, batch requests, aggregate user data, manage connections, and cache data and requests."
- "To *Reduce costs* one can segment the workload and cache data and requests."

However, they also make it clear that the validity of these techniques relies heavily on special hardware in order to guarantee scalability, and as such they only offer limited scalability. They proceed to state that "observations have led us to confirm that implementing full scalability is not an obvious task in software development, and especially with conventional approaches". Fayad et al. also talks about something called *"Horizontal Scalability"*, which "emphasizes the ability of the architecture to extend their boundaries by establishing connections with other software architectures in an efficient manner". They further state that horizontal scalability can be achieved in two directions, referred to as *Scaling Out* and *Scaling In*, also called *Extensibility* and *Reduction*. Extensibility is defined as the capacity of architectures to bind external architectures (called "leaves") to their structure, and by doing so "creating a synergy between these dissimilar architectures". Reduction is defined as the process of unbinding those external architectures from the structure without causing harm.

Häggander and Lundberg[17] states 10 guidelines which they feel are relevant to consider when designing large real-time applications to successfully establish a balance between performance, maintainability and flexibility, all which are key elements in scalability.

They state the following guidelines:

1. "Consider a multiprocessor when the supposed application design consists of many independent system instances."

2. "Consider a multiprocessor when the supposed application design is based on third party systems which have shown themselves to scale-up well on multiprocessors."

3. "Reconsider using traditional performance optimization techniques instead of multiprocessors."

4. "Consider multiple processes for high performance and scalability first."

5. "Consider multithreaded programming in combination with multiple processes."

6. "Avoid frequent allocations and de-allocations of dynamic memory."

7. "Meet maintainability requirements with exchangeable components instead of extendable components."

8. "Evaluate a heap implementation optimized for multiprocessors."

9. "Utilize the stack memory if possible."

10. "Consider multiple multithreaded processes for maximal maintainability, flexibility and performance."

They do, however, emphasize that "it is not always possible to to maximize each quality attribute in a design, thereby making trade-offs necessary", and that one of the major challenges in software design is to find solutions that offers a good balance of these attributes.

### Research Question 1 and 3

*Research Question 1: How do one successfully develop a scalable distributed system? Which factors needs to be considered, and how does one achieve these? Without knowing what needs to be done, developing a high-grade scalable system may be difficult, and cost a lot of extra time.*

*Research Question 3: Which kind of systems can benefit from having a high-grade scalability? It is important to know if a certain system actually benefits from being highly scalable, and if so, how much. If the benefits are relatively small, there might be cheaper ways of developing it.*

Caruso [5] states that there are certain important things to consider when developing a scalable system that is to be capable of handling a wide range of load levels. He argues that the most important factor is "the concept of subdividing load into dynamically allocable, load-sharing fragments (subclients)", and that by doing this the system gains the ability to adjust to a wide range of different load/demand levels. He also mentions that one of the characteristics of a scalable system as being able to access existing data flows efficiently as well as creating new ones.

Lipson [3] argues that an important part of scalable systems is *modularity*. He defines modularity as being the ability to take a part of a system which isn't dependant on any parts outside of itself, which then is a module. If one then has several such modules, they can be used as ”building blocks” to create an application, where each module can be removed or exchanged independently. He states that while there are several drawbacks to modularity, most notably non-optimal performance and reuse. Non-optimal performance since a integrated system can be more efficient ”as information, energy and materials can be passed directly within the system”, and reuse since a module that fits in slot A might need to be adjusted to fit in slot B, which then might make it unusable in slot A. He argues, however, that modularity becomes justified in the long run thanks to the system becoming more adaptable.

Amdahl [18] states (in what now is commonly referred to as ”Amdahl's Law”) that there is a limit to the speedup a program can gain from parallel processing (which is an important factor in scalability). He means that there will always be a certain sequential overhead which cannot be parallelized, and thus cannot be made faster by using parallelization.

### Research Question 1.1

*Research Question 1.1: Which technologies, libraries and software can one use to achieve high-grade scalability? Not all technologies etc. support a high level of scalability, and knowing the difference between the ones who do and the ones who do not can save a lot of expenses.*

Gropp, Lusk, Doss and Skjellum[8] discusses the design choices made when designing the MPI standard and their popular implementation MPICH. They compare the cross platform standard that MPI is against a vendor specific predecessor, and the performance penalties the cross platform approach resulted in. But also why it was a good tradeoff since applications don't need to be rewritten when switching between computer systems. While the api towards the user of MPI is the same everywhere, MPICH still provides the ability to have platform specific code in MPI to speed up the execution time on different platforms. Already in 1996 when this article was written several vendors had contributed code to get MPICH to run faster on their hardware.

Miller [12][11] gives a hands on approach on how to write scalable application with python and MPI, and he also argues that even if a program written in C may run faster, the counterpart in python will be much easier to write and maintain, and therefore would be preferred in many cases. The performance in the python case could also be increased by writing heavy computing parts in C, or the python code could be used as a prototype for the real application. Miller gives detailed yet easy to understand explanation of the principle behind MPI, as well as how to write a parallel application with it.

### Research Question 3

*Research Question 3: Which kind of systems can benefit from having a high-grade scalability? It is important to know if a certain system actually benefits from being highly scalable, and if so, how much. If the benefits are relatively small, there might be cheaper ways of developing it.*

Jogalekar and Woodside [2] propose a number of methods for evaluating the scalability of a system. They mean that a system is scalable ”if it can be deployed effectively and economically over a range of different ”sizes””.

They state four different cases where their proposed scalability analysis can be of use:

- ”To make a strategic choice of a scaling strategy, among several alternatives.”
- ”To tune the available scaling enablers for achieving the most cost-effective system configuration at any given scale.”
- ”To assess the need for providing more scaling enablers.”
- ”To assess the need to re-engineer the system, given the available choices for an expected evolution of the system.”

# 7 Analysis and discussion

If we analyze the findings from the literature review we find that there are a number of things one should do in order to achieve full scalability. The system should be parallelized and, at least in a large-scale distributed system, somewhat modular. Parallelization should be done in order to segment the workload across several machines, which serves to both improve performance and generally increase system capacity[6][17][5]. This can be done using technology such as MPI[8][11][12] or similar.

This supports our own findings from developing and running a simple, parallelized password finder using MPI (section 4), which achieved significantly improved performance compared to a non-parallelized version of the application, with little difference in development time. The improved performance coupled with the relative ease of development suggests that parallelization is, if done right, a relatively low-cost, effective method for achieving scalability. However, it is important to mention that while the difference in development time cost between our applications (parallel and non-parallel) was insignificant, the difference will likely become larger in more complex system development.

Modularity is a way of making the system more adaptable by building the system of several modules, i.e. smaller pieces that does not have any external dependencies[3]. By making a system adaptable by modularity it gains the possibility to remove certain, non-vital functions, or add new ones, without harming the system. This can, in theory, enable it to deploy on a wide range of platforms with relatively little modifications. Modularity usually comes at the expense of slightly lesser performance and less reuse, but for widely distributed systems it might very well prove worth it in the end. To make an entire system modular is close to impossible, since keeping modules independent from each other is often not doable, but some modularity (and the adaptability gained from it) can prove to be an asset in certain systems, but our findings suggest that it is not vital in order to achieve scalability.

Our findings suggest that the cost for developing a scalable system varies a lot depending on the system. While development costs can be kept quite low if done right, there is also the question of hardware. In some of our findings the scalability issue is, in part, addressed by simply adding better hardware[6][17], thereby making the system able to handle larger loads. While the cost of hardware is often seen as insignificant compared to development costs, it is a cost nonetheless, and thus cannot be ignored. The cost of development is hard to measure, since, according to us, a system can never be *too scalable*, and thus an effort to make a system as scalable as possible may go on for a very long time. However, it might not always be worthwhile to make a system scalable beyond a certain point. As our findings suggest, there is a limit to how scalable a system can actually become[18], and so we feel that it is important to have a clear understanding, from the start, of how much effort should be put into making the system scalable.

## 8 Conclusion

Based on our findings, we believe that there are a couple of points necessary to consider when considering scalable development.

- **Will the system benefit from this?** If it does not, scalability might not be worth the extra effort and cost. It is important to try to calculate how high the system load will be, and how much it needs to be able to handle in extreme cases. Sometimes, if the extreme cases doesn't differ too much from the ordinary, it can be enough by just adding faster hardware, which tends to be cheaper than the extra development scalability requires. However, it is important to be meticulous when making this decision, since modifying a non-scalable system into becoming scalable can require a lot of work, which can become costly.

- **How do one best make a system scalable?** We have presented one technology which is viable to use (MPI), but there are many more, and there is no one, general solution that works for all systems. Therefore it is important to investigate which solution and technology that works best for a specific system.

- **How scalable should the system be, and how much time should be assigned to the development of it?** This varies, once again, a lot from system to system. If one anticipates a large system load, putting more effort into scalability should be a priority, especially if the load fluctuates from high to low on a regular basis. However, if the system load increases in a more linear way, it might not be necessary to put as much effort into scalability. It is also important to remember that a system cannot effectively scale beyond a certain point, which can easily make investing a large amount of time and resources into scalable development a waste.

# References

[1] J.S. Rellermeyer and G. Alonso. Services everywhere: Osgi in distributed environments. *EclipseCon 2007*, 2007.

[2] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 11(6):589–603, 2000.

[3] H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. *Journal of Biological Physics and Chemistry*, 7(4):125, 2007.

[4] M. Van Steen, S. Van der Zijden, and H.J. Sips. Software engineering for the scalable distributed applications. In *Computer Software and Applications Conference, 1998. COMPSAC'98. Proceedings. The Twenty-Second Annual International*, pages 285–292. IEEE, 1998.

[5] Caruso J.A. Toward a scalable design for command and control systems. *Parallel and Distributed Real-Time Systems, 1997. Proceedings of the Joint Workshop on*, pages 43–52, 1997.

[6] M.E. Fayad, H.S. Hamza, and H.A. Sanchez. Towards scalable and adaptable software architectures. In *Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on.*, pages 102–107. IEEE.

[7] C.C. Wu, L.F. Lai, and Y.S. Chang. Using mpi to execute a fuzzyclips application in parallel in heterogeneous computing systems. In *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*, pages 279–284. IEEE, 2008.

[8] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[9] What are alternatives to MPI for communication in parallel computing. `http://insidehpc.com/2006/06/19/what-are-alternatives-to-mpi-for-communication-in-parallel-computing/`.

[10] V. Getov, S.F. Hummel, and S. Mintchev. High-performance parallel programming in java: Exploiting native libraries. *Concurrency - Practice and Experience*, 10(11-13):863–872, 1998.

[11] P. Miller. pyMPI–An introduction to parallel Python using MPI. *Livermore National Laboratories, Jan*, 2002.

[12] P. Miller. Parallel, distributed scripting with python. In *Third Linux Clusters Inst. Int'l Conf. Linux Clusters: The HPC Revolution.*

[13] T.J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, pages 308–320, 1976.

[14] Boost.MPI Python Bindings. `http://http://www.boost.org/doc/libs/1_46_1/doc/html/mpi/python.html`.

[15] Python Mpi test code. `http://github.com/Norberg/pympi-test`.

[16] PyMetrics produces metrics for Python programs. `http://sourceforge.net/projects/pymetrics/`.

[17] D. Häggander and L. Lundberg. Ten Performance Guidelines for Balancing Software Quality Attributes when Developing Large Real-time Applications for Multiprocessors. *Technical Report*, 99(16), 1999.

[18] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

# A Selected papers for Literature Review

During the process of reviewing the found papers we removed three that we felt did not, while being relevant, contribute to our cause, and thus they were removed.

The ones we selected are mentioned here:

**Evaluating the scalability of distributed systems** [2]

*A system design is scalable if it can be economically deployed at a range of scales, in both small and large con- figurations. Little attention has been paid to measuring and comparing the scalability of different designs of software for distributed operation. Recently a new measure of scalability (called here P-scalability since it based on the "power" metric) has been defined specifically for distributed systems. This paper generalizes the metric, defines a scaling path embodying a strategy modifying the system as it is scaled up, and employs scalability enabling parameters which adapt the system to give the maximum value of the scalability metric, at any point along the scaling path.*

**Principles of modularity, regularity and hierarchy for scalable systems** [3]

*Scalability of open-ended evolutionary processes depends on their ability to exploit functional modularity, structural regularity and hierarchy. This paper offers a number of observations about properties, dependencies and tradeoffs among these principles and proposes a formal model where such ele- ments can be examined.*

**Software engineering for the scalable distributed applications** [4]

*A major problem in the development of distributed applications is that one cannot assume that the environment in which the application is to operate will remain the same. This means that developers must take into account that the application should be easy to adapt, A requirement that is often formulated imprecisely is that an application should be scalable. The authors concentrate on scalability as a requirement for distributed applications, what it actually means, and how it can be taken into account during system design and implementation. They present a framework in which scalability requirements can be formulated precisely. In addition, they present an approach by which scalability can be taken into account during application development. Their approach consists of an engineering method for distributing functionality, combined with an object-based implementation framework for applying scaling techniques such as replication and caching.*

**Toward a scalable design for command and control systems** [5]

*Command and control systems exist in a world of con- stantly shifing demands and expectations: therefore, pro- ducing scalable, evolvable systems has* *always been a priority of system designers. Howevel; the need to meet peiformance requirements has ofen been a restraint on scalability goals. Advances in the speed and capacity of computational and communications equipment have sub- stantially increased the range of scalability that can be achieved while still meeting performunce requirements. The High Performance Distributed Computing Program (HiPer-D) has demonstrated a design for the AEGIS weapons system, which is close to achieving the full range of operational requirements while providing scalability along several dimensions. This paper discusses the HiPer-D design, its context, development, and efforts to achieve system level performance.*

**Towards Scalable and Adaptable Software Architectures** [6]

*Developing scalable and adaptable architectures that can accommodate evolving changes is crucial for reducing software development cost. To achieve scalability and adaptability, developers should be able to identify where and how new (current) layers will be added (removed) from the architecture. Failing to do so may lead to software architectures that require a considerable modifications when the system evolves or changes due to new or added requirements. In this paper, we address the problem of developing scalable software architectures that can accommodate new and/or modified requirements without the need for re-developing the architecture from scratch. The approach is demonstrated through a case study.*

**A high-performance, portable implementation of the MPI message passing interface standard** [8]

*MPI (Message Passing Interface) is a specification for a standard library for message passing that was defined by the MPI Forum, a broadly based group of parallel computer vendors, library writers, and applications specialists. Multiple implementations of MPI have been developed. In this paper, we describe MPICH, unique among existing implementations in its design goal of combining portability with high performance. We document its portability and performance and describe the architecture by which these features are simultaneously achieved. We also discuss the set of tools that accompany the free distribution of MPICH, which constitute the beginnings of a portable parallel programming environment. A project of this scope inevitably imparts lessons about parallel computing, the specification being followed, the current hardware and software environment for parallel computing, and project management; we describe those we have learned. Finally, we discuss future developments for MPICH, including those necessary to accommodate extensions to the MPI Standard now being contemplated by the MPI Forum.*

**pyMPI - An introduction to parallel Python using MPI** [11]

*The interpreted language, Python, provides a good framework for building scripts and control frameworks. While Python has a (co-routining) thread model, its basic design is not particularly appropriate for parallel programming. The* *pyMPI extension set is designed to provide parallel operations for Python on distributed, parallel machines using MPI.*

**Parallel, distributed scripting with python** [12]

*Parallel computers used to be, for the most part, one-of-a-kind systems which were extremely difficult to program portably. With SMP architectures, the advent of the POSIX thread API and OpenMP gave developers ways to portably exploit on-the-box shared memory parallelism. Since these architectures didn't scale cost-effectively, distributed memory clusters were developed. The associated MPI message passing libraries gave these systems a portable paradigm too. Having programmers effectively use this paradigm isa somewhat different question. Distributed data has to be explicitly transported via the messaging systemin order for it to be useful. In high level languages, the MPI library gives access to data distribution routines in C, C++, and FORTRAN. But we need more than that. Many reasonable and common tasks are best done in (or as extensions to) scripting languages. Consider sysadm tools such as password crackers, file purgers, etc... These are simple to write in a scripting language such as Python (an open source, portable, and freely available interpreter). But these tasks beg to be done in parallel. Consider a password checker that checks an encrypted password against a 25,000 word dictionary. This can take around 10 seconds in Python(6 seconds in C). It is trivial to parallelize if you can distribute the information and co-ordinate the work.*

**Ten Performance Guidelines for Balancing Software Quality Attributes when Developing Large Real-time Applications for Multiprocessors** [17]

*A simple and cost-effective way to obtain high performance in large real-time applications is to utilize high performance hardware. Symmetric Multi-Processors (SMP:s) are today the mainstream in high performance hardware. SMP is a parallel hardware architecture and thus relies on a parallel application software to operate efficiently. A strong focus on quality attributes such as maintainability and flexibility has, however, resulted in a number of new methodologies, e.g. object oriented design and third party class libraries, which when incorrectly applied significantly limit the application performance. Parallel applications have proved to be more vulnerable in this respect than sequential ones. It is thus not always possible to maximize each quality attribute in a design, thereby making trade-offs necessary. A major challenge is thus to find solutions that balance and optimize the quality attributes, e.g. SMP performance contra maintainability and flexibility. We have studied three large real-time telecommunication applications developed by the Ericsson company. In all these applications maintainability and flexibility are strongly prioritized. The applications are also very demanding with respect to performance due to real-time requirements on throughput and response-time. SMP:s and multithreading are used in order to give these applications a high and scalable performance. Our result is presented in form of 10 guidelines assembled with the aim of helping designers* *of large real-time applications to establish balance between SMP performance, maintainability and flexibility.*