A Scalable Entry-Level Architecture for Computational Grids based on Web Services

Mario Jeckle, Ingo Melzer, and Jens Peter

University of Applied Sciences Furtwangen
Robert-Gerwig-Platz 1, D-78120 Furtwangen, Germany
mario@jeckle.de, paper@ingo-melzer.de, info@jens-peter.com
http://www.jeckle.de/ http://www.ingo-melzer.de/

Abstract. Grid computing has recently become very popular and this development deserves to be called a hype. To benefit from the techniques of Grid computing, it is not necessary to invest a lot in software for smaller solutions. Simple Grids sized at entry-level can be implemented using some ideas of service-oriented architectures. Especially when building such a new system, it is interesting to know, how the result will most likely perform and how big the benefits will be.
This paper gives an approach to implement such an entry-level Grid solution for computational tasks and introduces a computational model to estimate the performance of these solutions.

1 Introduction

At the latest, since the Grid project *SETI@home* [7] became well known, Grid computing can be called a hype and many software companies try to benefit from this development. However, for smaller system a Grid solution can easily be implemented from scratch, and platform independent solutions can be applied in most system environments. Based on the idea of service-oriented architectures, a simple and flexible approach is presented in this paper for implementing computational Grids. Also a computational model is introduced to estimate the performance of Grid solutions.

The remainder of this paper is structured as follows. First, the young history of Grid applications is explored and different instances of Grids are sketched. Based on this, the paradigm of Service-oriented architectures which is increasingly confluent with Grid-based techniques is introduced. Additionally, interpreted programming languages with platform independent execution environments are discussed since these languages seem to be well suited for implementing Grid contributing nodes.

Based on this, section three introduces a new computational model which allows the estimation of expected performance for Grid applications relying on the techniques introduced before. The computational model refactors proven heuristics well-known in the field of parallel processing. As a foundation of our proposed formula two prototypical Grid implementations sketching edge cases are discussed.

2 Technologies Involved

2.1 Grid Computing

The Grid – A brief history Historically, the name *Grid* was chosen by virture of the analogy to the power grid, which distributes electrical power to each citizens power outlet, without knowledge where the electricity came from. Likewise, the vision of applying the grid idea to computing is that computing power (i.e., the ability to store or process data) will be available also through an outlet. When aranging the evolution of Grid into generations, the focus can be set on the standardization of the technology involved.

The pre-history of applying the grid idea to computing is based on technologies for distributed computing like the Common Object Request Broker Architecture (CORBA), which is standardized by the Object Management Group [8]. Also techniques like Remote Method Invocation (RMI) and JINI [9] from Sun were launched to provide a software infrastructure for distributed computing. And also DCE and DCOM are proposed as solutions for Grid-computing.
All these technical approaches share the characteristc of beeing a *point solution* to the issues to be addressed when stiving a solution for interconnecting various resources. The term point solution (which was initially coined by [5]) here referes to techniques which may contribute to solve certain aspects of the main problem but which fail in addressing all issues at one time.

In the early 1990s some research projects focused on distributed computing were founded. The publication of early results to the I-Way [5] project which were presented at the 1995 Super Computer conference represent the first true Grid and thus mark the first generation of such approaches. The I-Way implementation connected 17 high-end computers over high-performance networks to one *metacomputer*, which runs 60 different applications. The success of this demonstration led the DARPA fund a new research project titled Globus [4] under the lead of Foster and Kesselmann.

Another project which fits in the first generation of Grid computing is Factoring via Network-Enabled Recursion (FAFNER for short) [6], which was launched in context of the *RSA Factoring Challenge*. FAFNER creates a metacomputer which was deployed to attack content that is cryptographically secured by using the RSA algorithm. In detail FAFNER strives to attack the basic idea of the RSA algorithm, which is the usage of large composite numbers for securing arbitrary content. Since fast factorization or large composite numbers is a challenge which still lacks an efficient mathematical algorithm, attacks require enormous amounts CPU time. Technically speaking, FAFNER provides a Web interface for a factoring method which is well suited for parallel and distributed execution, the *number field sieve* [19]. The project implemented a software daemon which is based on the Perl scripting language which are accessed through the Common Gateway Interface protocol. The daemon handles the retrieval of input values to the local system and the submission of results

via HTTP's well-known GET and POST methods. This approach proved to be successful and paved the way of other Web-based projects.

Other, mostly scientific based projects, like SETI@HOME [7], GENOME@HOME, or FIGHTAIDS@HOME were launched. Also some mathematic based problems gained help from concentrated computing power like the GREAT INTERNET MERSENNE PRIME SEARCH or the GENERALIZED FERMAT PRIME SEARCH. The still increasing number of computers connected to the internet via its access to the World Wide Web widens the amount of potential contributors to these projects.

With the current, i.e. the second, generation of the Grid, three main issues have to be focused. These were heterogeneity, scalability and adaptability. In this context answers to the following questions have to be found:

- **Identity and Authentication**: How should machines which are authorized to participate within the Grid be uniquely identified and authenticated?
- **Authorization and Policy**: How is decentralized authorization handled and how are certain policies (like Quality of Service) guaranteed?
- **Resource Discovery**: How are resources which are offered by the Grid discovered?
- **Resource Allocation**: How is handling of exclusive or shared resource allocation handled?
- **Resource Management and Security**: How are the distributed resources managed in an uniform way, esp. how can a certain level of security be provided for all nodes participating within the Grid?

Today there are enormous efforts of companies like IBM, Sun, and HP to advance the Grid idea to a next evolutionary step. The main focus here is the service-oriented architecture approach on which the next generation of Grids will be based. In essence the idea of distributed processing and data storage underlying the Grid idea is increasingly merged with the technical infrastructure summarized as WEB SERVICE. By doing so the level of standardization reached by virtue of widely accepted Grid infrastructures such as the GLOBUS toolkit is leveraged by the ubiquitous technical infrastrucutre of Web-based services.

Data Grid vs. Computational Grid The current state of applying the Grid idea is summarized by [4] as:

> "Grid computing enables virtual organizations to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control, omniscience, and existing trust relationship"

The absence of such a central control means, that there is no centralized instance acting like an operating system which is in power to manage execution characteristics (e.g. scheduling, memory management, interrupt handling). But, for ease of management purposes some Grid implementations of major software

vendors deploy a centralized control in terms of a dedicated node offering the tasks to be computed to the participating Grid nodes.

The abstract notion of *Grid* is differentiated further by companies providing software and hardware infrastructure for Grids like IBM and Sun. In detail Sun refers to *CampusGrid* while IBM chooses the term *IntraGrid* when addressing Grids which are deployed company internal only.
Consequentially, company-spanning Grids are termed *ExtraGrids* and Grid-based solutions relying on the Internet's technology are named *InterGrids*. Since the latter ones are potentially distributed globally Sun introduced the additional term *GlobalGrid* [11].

Further, often Grids are distinguished from an operational point of view. Here two major directions have emerged. First, data-centric Grids which focus on the transparent access to data distributed geographically. In essence these "Grids should provide transparent, secure, high-performance access to federated data sets across administrative domains and organizations"[10].

An example for a data Grid is the *European DataGrid Initiative* [12], a project under the lead of CERN, the European nuclear research center. CERN's data Grid initiative which also involves IBM should manage and provide access from any location worldwide to the unpresented torrent of data, billions of gigabyte a year, that CERN's *Large Hadron Collider* is expected to produce when it goes online in 2007 [13]. The second mainstream of Grid computing is the computational Grid. In this approach, the idle time of network detached machines will be shared and used by other Grid-enabled applications. A computational Grid can be based on an *IntraGrid*, which uses the spare CPU-time of desktop computers for compute intensive tasks. Such a systems is used by the swiss company NOVARTIS to help solving problems in the medical research. The systems uses 2700 desktop computers to provide a computing power from about 5 terra FLOPS [14].

In this paper we describe an architecture which combines the technical aspects (i.e., the deployment of Web-based service-oriented technology) of InterGrids with the application area of IntraGrids. In detail this means benefiting from the transfer of techniques typically found in InterGrids to closed organizations which deploy IntraGrids to Grids which are deployed accross organizational boundaries. The advantage by doing so lies in lowered costs of infrastructure and an increased amount of standardized components. Additionally, our approach takes this further and establishes the notion of an entry-level Grid which adds ease of deployment and operation.

2.2 Service-oriented Architectures

Some of the latest Internet-related developments share the idea of utilizing different functionalities over the net without the requirement of using a browser. The most general and to some degree visionary version is termed SERVICE-ORIENTED ARCHITECTURE, or for short SOA.

The basic idea of a SOA is quite simple. A developer implements a service, which can be any functionality made available to others, and registers his service in some global registry like some yellow pages for services. A user, which is most often some other service, is thereafter able to find the registered service, retrieve all information about invoking the just found service, and call the service. In the end, all this should happen without human involvement. This last step is called *loosely coupled*, because the connection or coupling is made at run-time when needed, in other words just in time.

2.3 Web Services

Today, the most advanced instance of a SOA is called Web services. The technical basis of the Web service philosophy is grounded on the idea of enabling various systems to exchange structured information in a decentralized, distributed environment dynamically forming a extremely loosely coupled system. In essence this lead to the definition of lightweight platform-independent protocols for synchronous remote procedure calls as well as asynchronous document exchange using XML encoding via well-known Internet protocols such as HTTP.

After some introductory approaches which were popularized under the name XML-RPC [22] the SOAP[1] protocol which has been standardized by the World Wide Web Consortium [1, 2] establishes a transport-protocol agnostic framework for Web services that could be extended by the user on the basis of XML techniques.

The SOAP protocol consists of two integral parts: A messaging framework defining how to encode and send messages. And an extensibility model for extending this framework by its user. Firstly, a brief introduction of the messaging framework is given before showing value of the extensibility mechanisms to accomplish the goals defined above.

Technically speaking, SOAP resides in the protocol stack above a physical wire protocol such as HTTP, FTP, or TCP. Although the specification does not limit SOAP to HTTP-based transfers, this protocol binding is currently the most prominent one and is widely used for Web service access. But it should be noted that the approach introduced by this paper is designed to operate completely independent of the chosen transport protocol and resides solely on the SOAP layer.

All application data intended to be sent over a network using the SOAP protocol must be transferred into an XML representation. To accomplish this, SOAP defines two message encoding styles. Therefore, the specification introduces rules for encoding arbitrary graphs into XML. Most prominent specialization of this approach is the *RPC style* introduced by the specification itself which allows the exchange of messages that map conveniently to definitions and invocations of method and procedure calls in commonly used programming languages. As introduced before SOAP is by nature protocol agnostic and can be deployed for

[1] At the time of its definition the acronym stood for *Simple Object Access Protocol*. In the standardized version SOAP is no longer an acronym.

message exchange using a variety of underlying protocols. Therefore a formal set of rules for carrying a SOAP message within or on top of another protocol needs to be defined for every respective transport protocol. This is done by the official SOAP specification for HTTP as well as SMTP.

Inside the SOAP protocol the classical pattern of a message body carrying the payload and an encapsulating envelope containing some descriptive data and metainformation is retained. Additionally SOAP allows the extension of the header content by the use of XML elements not defined by the SOAP specification itself. For distinguishing these elements from those predefined by the specification the user has to take care that they are located in a different XML namespace.

The example below shows a complete SOAP message accompanied with the transport protocol specific data necessary when using the HTTP binding. Additionally a user defined header residing in a non-W3C and thus non normative namespace is shown as part of the SOAP `Header` element.

```
POST /axis/theService/ HTTP/1.1 Content-Type: text/xml;
charset=utf-8 Accept: application/soap+xml
Host: 10.0.0.1:8080
Content-Length: nnn

<?xml version="1.0" ?>
<env:Envelope
 xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
    <ns1:DeliveryNotification
    env:mustUnderstand="true"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"
       xmlns:ns1="urn:xmlns:daimlerchrysler.com:research">
       <ns1:SendTo URI="MailTo:john.doe@daimlerchrysler.com"/>
    </n1:DeliveryNotification>
 </env:Header>
 <env:Body>
    <ns2:CalcParams>
       <ns2:ID>7492653</ns2:ID>
       <ns2:x>42</ns2:x>
       <ns2:y>3.14</ns2:y>
       <!-- more details omitted for brevity ... -->
    </ns2:CalcParams>
 </env:Body>
</env:Envelope>
```

In contrast to the payload which is intended to be sent to the receiver of the SOAP message clearly identified by HTTP's `Host` header, SOAP headers may or may not be created for processing by the ultimate receiver. Specifically, they are only processed by machines identified by the predefined `role` attribute. By doing so, the extension framework offers the possibility of partly processing a message along its path from the sender to the ultimate receiver. These intermediate processing steps could fulfill arbitrary task ranging from problem oriented ones like reformatting, preprocessing, or even fulfilling parts of the requests to more infrastructural services such as filtering, caching, or transaction handling. In all cases the presence of a node capable of (specification compliant) processing of a SOAP message is prescribed. This is especially true since an intermediary addressed by the `role` attribute is required to remove the processed header after executing the requested task. Additionally, the specification distinguishes between headers optionally to be processed (e. g. caching) and those which are interspersed to trigger necessary message behavior. The latter ones must additionally be equipped with the attribute `mustUnderstand`. If a header addressed to an intermediary flagged by this attribute cannot be processed, the SOAP

node is forced to raise an exception and resubmit the message to the sender. Thus it is ensured that all headers mandatory to be processed are consumed by the respective addressees and removed afterwards.

Loosely Coupled An important property of a SOA and Web services it the fact that they are loosely coupled. This means that they are not statically liked and binding does not happen at compile time. During its execution, a service can search for some other services, which might at this moment in time still be unknown, retrieve information about the search results, and invoke one of the just found services.

This allows to move services to different machines and simple change the information in one of the registries. No other service has to be changed or recompiled. A promising starting point for a highly flexible infrastructure.

WS-Notification Later specifications such as WS-Notification [24] allow the implementation of the publish/subscribe pattern. This allows to automatically trigger certain action as soon as certain criteria have been met.

WS-Notification allows the implementation of Grid infrastructures and Grid based applications. The Open Grid Services Architecture, short OGSA, moves very close to the Web services world and the latest version is based on Web services standards.

If this development continues, it will soon be very difficult or even impossible to tell Web services and Grid services apart. This development became obvious one year ago in the Open Grid Services Infrastructure, short OGSI, definition 1.0 which has been released in April 2003. It states in this context "a Grid service is a Web service that conforms to a set of conventions (interfaces and behaviors) that define how a client interacts with a Grid service".

There are a number of alternatives to WS-Notification. Some features can be implemented using WS-Eventing [23] and part 2 of WSDL 2.0 offers a message exchange pattern, too.

2.4 Interpreted Languages

One of the main challenges for Grid application frameworks such as the OGSA/OGSI approach is multi-platform support. That is for avoiding prescriptions concerning platform details such as the deployed operating system running machines participating the Grid. Also the infrastructure should not set out limits regarding the programming language chosen for implementing machine's code locally executed but contributing to the Grid.

In general these challenges can be tackled by providing portable implementations as well as by specifying solely interfaces whose description do not reveal details of the programing language specific manifestation. Classically, the required interfaces descriptions are provided by using standardized paradigm neutral approaches such as CORBA's Interface Definition Language (IDL) or WSDL, the in some sense comparable approach introduced by Web service technology.

Unfortunately, description mechanisms which concentrate solely on callable interface do specifiy wire representations of the data to be exchanged between communicating nodes. Therefore typically extra agreements have to be settled. In CORBA's case this is the Common Data Representation resp. the XML for Web service deployment.

Besides this the internal implementation of nodes participating the Grid may vary vastly. In general this is rather a blessing than a cure. But, especially for entry level Grids the costly creation or even adaptation of code to deploy on single nodes should be as lightweight as possible. Likewise, the potential requirement to port applications to other platforms curbs the amount of specialities of the respective language platform which can be used to an absolute minimum. Therefore the reliance on the lowest common denominator, i.e. basic functionality known to be supported by various plattforms is an archetypical design decision to ensure widest possible portability.

Our approach for supporting entry-level Grids therefore introduces the usage of a portable implementation and execution plattform as third cornerstone (besides basic Grid ideas and reliance on standardized Web service technology) of the proposed architecture.

It should be emphasized that this does not tie the approach to a single specific platform. But it has not escapted our notice thas this founds a clear preference for interpreted or at least hybrid (i.e. approaches which incorporate a compile cycle which produces a result which is interpreted at runtime) languages such as formed by the JAVA or Microsoft .NET platform.

As a result of this architectural constraint we are able to interconect various physical platforms on short notice to a metacomputer. The computational nodes constituting the metacomputer will be able to offer current programing paradigms such as object orientation and concurrency without additional adaptation effors. Additionally, basing Grid applications on current programming approaches bears twofolded benefits for both, the Grid and the deployed software execution environment. On the one hand deployment of Grid-based technology is leveraged by the level of additional standardization. On the the other hand the installation basis of the respective language environments is additionally widened.

3 Performance Estimation

3.1 Computation Model

The architecture presented in this paper is based on a Web service enabled server using the standardized SOAP communication protocol. The server hosts the main controlling service and a Web service toolkit. Due to the usage of standards it is technically feasible to add additional Grid nodes on short notice. These nodes share their local resources for processing by using the same software implementation available for different platforms and operating systems.

The main advantage of such an architecture is that the creation of a Grid is a lightweight process. In essence it simply requires the distribution of the software

and its install on the participating machines. At runtime these machines will act as clients requesting computational tasks fromt he node acting as server. At development time the application implementing the problem to be solved has to be transfered in an Grid-enabled application, so it is necessary to identify the code needed to build a *job list* which contains the amounts of work which should be distributed to the clients. Technically, one Grid node provide a *job queue* as a central service.

The next step is to wrap a Web service around the processing code of the application and deploy the service, which is an Web service RPC, on the server. To add new nodes to the Grid, solely the Web service has to be deployed to a network attached computer. Additionally, the new node has to be announced to the queue holding serving. This could be done by simply adding its network address to a configuration file or online by sending a predefined message.

This Grid is build up as a logical star topology, a single computer, the controller, coordinates the available resources. This controller has to fulfill different tasks in the Grid, for example he has to build the list of jobs, what means that the original task the Grid-enabled application has to process must be split in smaller pieces, the jobs, for parallel execution. The controller has to distribute these jobs to the available nodes, the computers which share their computing resources to the Grid and receive the processed results from the nodes. Communication with the participating nodes is operated in an asynchronous mode. Thus either asynchronous Web service calls have to be used or asynchronous communication has to be emulated on-top of synchronous communication. One way to achieve the latter ist to deploy multithreading within the controller.

On the node, there is also a Web service deployed which receives and process a job. The controller and the nodes are connected through a network, no matter if it is a local departemental network or the internet.

Fig. 1 is a schematically presentation of the architecture including the emulation of asynchronous communication by multithreaded synchronous communication.

1. The primary task is split into a number of single jobs by the *JobScheduler*.
2. The controller invoke for every active node an own *ServiceThread*.
3. Every thread grabs a jobs from the *JobQueue*.
4. The job is send to the nodes for processing and the *ServiceThread* wait for the result. When an error occurs, for example the Grid node is temporarily not available, the job is republished to the *JobQueue* and the *ServiceThread* will wait for an estimated time to send a new job to the node.
5. After the successful processing of a job, the node sends back the result to his *ServiceThread* and it is stored by the controller.

Based on that architecture, the following answers for the basic questions in Chapter 2.1 can be presented:

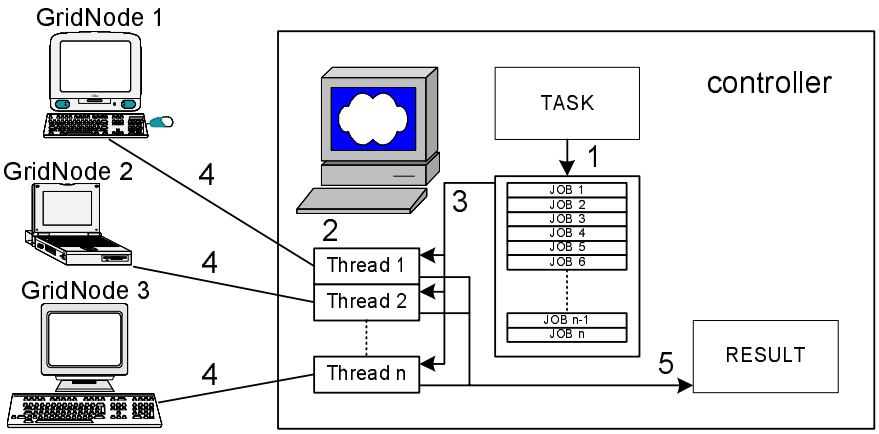

Fig. 1. Architecture

- **Identity and Authentication:** Solely the controller initiates connections to the nodes. The participating nodes can register themselves actively to take part within a computation task. For authenticating nodes the initiating SOAP requests can be signed digitally.
- **Authorization and Policy:** Based on a validated digital signature nodes are authorized to receive data for fulfilling the computation tasks. Policies concerning aspects of Quality of Service are ensured by the controller. If a node does not send back its computed result data in a certain amount of time the node is marked as inactive and the data is distributed to another node.
- **Resource Allocation:** Grid nodes can determine the share of processing time they devote to the computation of received tasks independent from the controller. In case of high node utilization the nodes are also allowed to withdraw from the Grid without requiring them to signal this to the controller.
- **Resource Management and Security:** Due to the usage of an interpreted language which can be executed on various platforms all resources can be handled in a uniform manner by utilizing the abstraction introduced by the execution platform. Security concerns can be addressed by deploying XML and Web service security mechanisms.

Within this model the following issues take influence to the over all performance of such a Grid:

Implementation of the Grid-enabled application: How much time is needed by the application to process the code which is not influenced by the parallel processing? Is there a *JobQueue* which contains the jobs prepared by a *JobScheduler* or must the controller extract every single job from the originally task at runtime? What happens with the results? What amount of

data need to be transfered for one job? How many jobs need to be processed in a second?

Performance of the controller computer: Because the controller has to coordinate every job and result, the performance of the controller can be a key indicator for the performance of the whole Grid. How many jobs per second can be transfered? Is there a local database the results where stored in? Must the controller host other applications than the Grid?

Networkperformance: With what type of network are the nodes connected to the controller? What is the maximal bandwidth of the network? Is every node in the same subnet? What is about the other traffic on the network?

Processingroutines on the Grid node: How much time need a node to process a job?

It is very interesting to forecast the performance benefit before transferring an application in a Grid-enabled application. There were some efforts to forecast the speedup of an application when running it on a parallel processor system. One of the formulas to forecast the speedup is the GUSTAFSON-BARSIS (1) approach:

$$SpeedUp = \#CPU + (1 - \#CPU) * P_{seq} \tag{1}$$

But this could not easy be transfered to a Grid where the parallel processors are connected via a network and the application and not the operating system need to handle the distribution of the processes to the additional computing power. In this case, not only the sequential (code which could not processed parallel) part of the application is an indicator for the speedup. Other parameters are the network latency, the time need to transmit the application parameters to process to the nodes, and the overhead the application produce to prepare sending and receiving of the parameters. By now, there is no approach to forecast such a speed up for Grid services.

In consideration of this issues the following formula (2) can give a forecast on the speedup of an entry-level Grid:

$$SpeedUp = \#CPU - \#CPU * (P_{seq} + NL) - \underbrace{(\#CPU * P_{over})}_{\text{GridFactor}} \tag{2}$$

The maximal speedup must be less than the number of processors so there are some other parameter we have to look at. The factor $\mathtt{P}_{seq}$ represents the sequential part of the application and will influence the maximum speedup because this factor will rise when the over all time the application need to process will decrease when it is processed by more computers. The network latency, or `NL`, is not static, it represents a range between 0 and 1 and has to be explored by a statistical review based on network performance tests.

A significant factor is $\mathtt{P}_{over}$, the overhead of performance of the application. This overhead and the number of used CPUs or Grid nodes is the so called *GridFactor*. This *GridFactor* is a key performance indicator for the Grid. Some

problems can be easy adapted for parallel processing but there is an immense overhead when there are more processing units involved. For example the processing time for a job is under 1 ms but the time needed to prepare the parameter for this job is about 100 times higher, there is no sense to process this kind of problem in a parallel way. With every additional Grid node the performance will go from bad to worse and with such a bad *GridFactor* there will be no speedup at all. The value of $\mathtt{P}_{over}$ has also be explored by statistical review based on tests with a prototype of the Grid-enabled application.

3.2 Validation

The results with a huge $\mathtt{P}_{over}$ and therefrom a high *GridFactor* can be shown with the Grid-enabled example application for matrices multiplication. The results of the time measurement tests with this application shows, that the processing time is stable, no matter if we have one node or ten nodes connected to the Grid. One handicap was that this application has had no *JobScheduler* to prepare the jobs and the other handicap was that the processing time on a nodes was approximate zero. The missing *JobScheduler* results in that the controller must prepare every job at runtime. And it take more time to prepare a job for a node than the node need to process it (about 1 nanosecond). The *GridFactor* for this kind of application is to high that there is no speedup at all.

The second application was implemented based on a thesis at the Fraunhofer Institute for Manufacturing Engineering and Automation IPA in Stuttgart [16]. It is a tool for analyzing networks, in this case specially to identify the leaks in the waver transport network of a computerchip fab. To enable the stand alone application for the Grid, the processing unit with the algorithms was extracted and the *JobScheduler* was modified. A Web service with the processing routine was deployed on the nodes and the amount of data was about 4kB for each job. Fig. 2 shows a sequence diagram of that application.

Fig. 3 shows the speedup reached with this application. Two different scenarios were tested with the application. The test were based on a sample network model with 31 nodes and 72 edges. Two different analyses were run on the model. The first test scenario was to analyze the network with one component to fail(*fault:1-test*). For this simulation 103 calculations have to be done. The second analyze simulates two defect components(*fault:2-test*) what results in 5253 calculations. The different speedups of the tests caused on the different $\mathtt{P}_{seq}$ and NL parts. In the *fault:1-test*, the ratio of $\mathtt{P}_{seq}$ and NL to the application time over all was higher. And the processing time on the clients a bit lower because the need to calculate only with one parameter. The *fault:2-test* shows a much better speed up. Within this test the ratio of $\mathtt{P}_{seq}$ and NL to the application time over all was lower, the time the nodes need to process a little bit higher.

With the Grid-enabled version of this software the time needed to analyze this small network with about 103 components shrink from over 80 minutes with a single slave to 6 minutes with 20 slaves. With the distributed computing it is

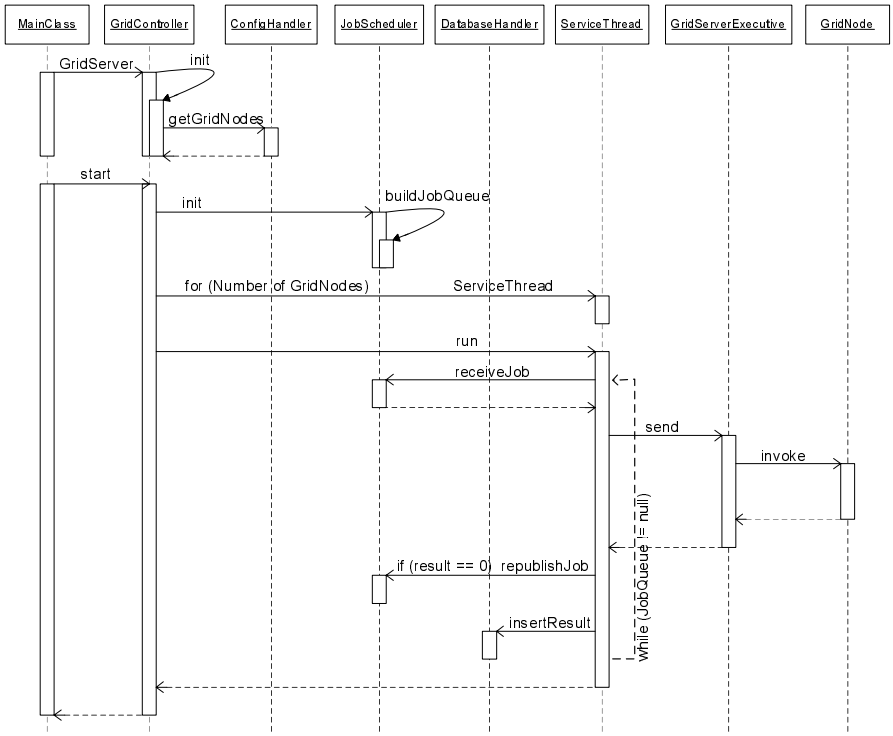

Fig. 2. Sequence Diagram

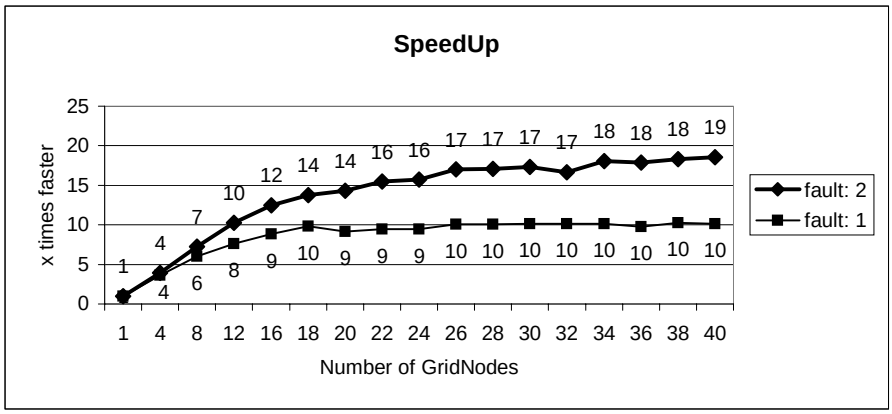

Fig. 3. SpeedUp

now possible to analyze a network in a manageable timeframe. The tests were executed on computers with a Athlon© 1100 MHz processor connected with a switched 100Mbit Ethernet. The Grid was based on a Tomcat server and the Web service framework Axis, both from the Apache Software Foundation [18].

The *GridFactor* for this application was better because the speedup is rising with the number of nodes. In Fig. 3 is shown that the speedup is rising slower with additional Grid nodes which is a result of the rising *GridFactor*. The tested application caused a processing time on the Grid nodes between 700 and 1100 milliseconds, so there was a average load from one call for every Grid node per second. Fig. 4 shows that there can be about 60 to 100 calls per second with a payload of 4kByte so `NL` is small and probably has not influenced the speedup stagnation happend by about 20 nodes.

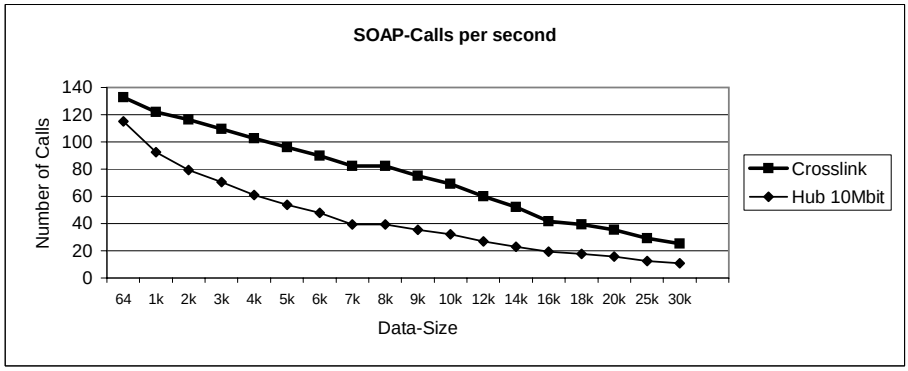

Fig. 4. SOAP-Calls per second

The maximum number of calls depending to the size of data (Fig. 4) was measured with the *SOAPing-Tool* [17], which can be used for testing the answer/reply behavior for a SOAP call in the tradition of the *ICMP Echo* service, better known as *PING*. The role of the controller fulfill a computer with a Pentium©-III Mobile processor with 1100 MHz and there was a computersystem based on an Athlon© XP 3000+ as a Grid node. The measured data is the average value from about 50000 *SOAPings* for each payload.

4 Related Work

Confluence of Web service technology and Grid applications are currently part of the next release of the OGSA/OGSI toolkit. It is expected by virtue of publications available from the Global Grid Forum that in future all nodes participating in a Grid (regardless is speaking of data or computational Grids) will

be accessible by using Web service interfaces. As a consequence the seminal standards WSDL and SOAP will widen its position as ubiquitous infrastructure based on the Web's base technology such as HTTP.

Concerning performance estimation of computational Grids only heuristics [20] and models with limited applicability [21] have been published so far. Both approaches addtionally lack consideration of entry-level technology such as interpreted languages and Web services.

5 Summary

It is not very difficult to implement platform independent solutions for computational Grids from scratch as it has been done for this paper. However, it is interesting to know how such solutions scale given the specific problem to be calculated. For this purpose, this paper has introduced a simple model including two formulas to estimate the performance of a computational Grid solution. This allows judge the potential benefit of an implementation before starting the real implementation and also helps to evaluate scenarios which might be used for Grid computing.

References

1. M. Gudgin, M. Hadley, J.-J. Moreau, H. F, Nielsen: W3C Candidate Recommendation: SOAP 1.2 Part 1: Messaging Framework, 20 December 2002 `http://www.w3.org/TR/soap12-part1/`
2. M. Gudgin, M. Hadley, J.-J. Moreau, H. F. Nielsen: W3C Candidate Recommendation: SOAP 1.2 Part 2: Adjuncts, 20 December 2002 `http://www.w3.org/TR/soap12-part2/`
3. R. Fielding, et al: Hypertext Transfer Protocol – HTTP/1.1, RFC2616, June 1999 `http://www.ietf.org/rfc/rfc2616.txt`
4. Globus `http://www.globus.org`
5. T. DeFanti, I. Foster, M. Papka, R. Stevens, T. Kuhfuss: *Overview of the I-Way.* International Journal of Supercomputer Applications, 10(2):123-130, 1996. `http://www.globus.org/research/papers.html`
6. FAFNER `http://www.npac.syr.edu/factoring.html`
7. SETI@home `http://setiathome.ssl.berkeley.edu/`
8. OMG `http://www.omg.org`
9. JINI `http://www.jini.org/`
10. A. Grimshaw: Data Grids in *Ahmar Abbas, Grid Computing: A Practical Guide to Technology and Applications*, 2004 Charles River Media, Hingham
11. A. Abbas: Grid Computing Technology - An Overview in *Ahmar Abbas, Grid Computing: A Practical Guide to Technology and Applications*, 2004 Charles River Media, Hingham
12. The DataGrid Project `http://eu-datagrid.web.cern.ch`
13. D. De Roure et al: The evolution of the Grid in *Fran Berman, Geoffrey C. Fox, Anthony J. G. Hey: Grid Computing*, 2003 John Wiley and Sons, West Sussex
14. Schlafende PC mutieren zum Supercomputer, Neue Zürcher Zeitung, 7. November 2003
    `http://www.nzz.ch/netzstoff/2003/2003.11.07-em-article97JEJ.html`

15. A Future e-Science Infrastructure
http://www.nesc.ac.uk/technical_papers/DavidDeRoure.etal.SemanticGrid.pdf
16. T. Geldner, *Graphenbasierte Layoutplanung von Transportnetzwerken in Halbleiterfabriken*, Konstanz/Stuttgart 2004
17. SOAPing http://www.jeckle.de/freeStuff/soaping/index.html
18. The Apache Software Foundation http://www.apache.org/
19. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard: The Number Field Sieve, ACM Symposium on Theory of Computing, 1990.
20. S. Sodhi: Automatically Constructing Performance Skeletons for use in Grid Resource Selection and Performance Estimation Frameworks, Proceedings of the 15th ACM/IEEE Supercomputing Conference, 2003.
21. C. Lee, J. Stephanek: On Future Global Grid Communication Performance, Global Grid Forum, 1999.
22. D. Winer: XML-RPC Specification, available electronically, 1999.
http://www.xmlrpc.com/spec
23. L. F. Cabrera, C. Critchley, G. Kakivaya et al.: WS-Eventing, available electronically, 2004.
http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf
24. S. Graham, P. Niblett, D. Chappell et al.: Web Service Base Notification, available electronically, 2004.
ftp://www6.software.ibm.com/software/developer/library/ws-notification/WS-BaseN.pdf
25. M. Gudgin, A. Lewis, J. Schlimmer (eds.): Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns, W3C Working Draft, World Wide Web Consortium, available electronically, 2004.
http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326

appendix. evtl. Code oder UML-Diagramme