Literature review

April 11, 2013

1 Introduction

In the literature review, we discuss the area of research covered by the thesis, citing relevant papers and articles that serve as a base of ideas for this document.
First, we talk about the Message-Passing Interface (MPI), the parallel programming API used for the purposes of this thesis. We also talk about the different implementations of this API, notably OpenMPI and MPICH. We discuss the methods of modeling and simulation in general. Then, we talk about SimGrid, which is a simulation-based framework, and SMPI, the implementation of MPI that runs on top of SimGrid. After that, we also describe StarMPI, a set of MPI communication routines, presenting a technique that can improve the performance of MPI applications.

2 MPI

Distributed computing is a very active and important subject of research in computer science, including fields such as cluster computing, grid computing, Cloud computing, or peer-to-peer computing. Communication between the different processes in a distributed application can be implemented in a number of ways. As communication is necessary in most cases, a standardized communication protocol can be a lot of help when developing a distributed program. The Message-Passing Interface (MPI) is a language-independent message-passing library interface specification. It is not a language, but a standard - there exist multiple MPI implementations. Since its take-off, it has become a de facto standard for inter-process communication. The standard provides vendors a clear set of routines, that they can implement efficiently, or in a way that it suits the hardware they provide.[3]

2.1 OpenMPI

OpenMPI is an MPI implementation with the goal of being able to achieve good performance on a wide range of different aspects of high-performance computing. To efficiently support multiple types of parallel machines, high performance “drivers” for all established interconnects are developed. These include TCP/IP, shared memory, Myrinet, Quadrics, and Infiniband. Features for checking data integrity are provided in order to account for network transmission errors. With the utilization of message fragmentation and striping over multiple (potentially heterogeneous) network devices, OpenMPI provides an increased bandwidth to applications, as well as the ability to handle the failure of network devices during runtime.[4]

2.2 MPICH

MPICH was originally developed during the MPI standards process starting in 1992 to provide feedback to the MPI Forum on implementation and usability issues. This original implementation was based on the Chameleon portability system to provide a light-weight implementation layer (hence the name MPICH from MPI over CHameleon). Around August 2001, development begun on a new implementation called MPICH2.[5] This implementation introduced improvements on collective communication operations by using multiple algorithms, choosing between them depending on certain variables - for example the message size.[6] Another important result during the development of MPICH2 was the design of the Nemesis communication subsystem and the porting of MPICH2 on that system. The efficient implementation of shared-memory communication helped Nemesis MPICH2 achieve low latency and high bandwidth.[7]
Starting with November 2012, the project is renamed to MPICH, with version number 3.0.[5]

3 Modelling and Simulation

In distributed computing, modelling means creating an abstraction of a real system by taking only the aspects of it that are relevant to the system's behavior into account. Once constructed, such a model becomes a tool with which we can investigate the behavior of the system.[8]

3.1 Advantages of Modeling

Modelling and simulation techniques have been used extensively in parallel computing and is an ongoing research topic, with new challenges continuously arising. There are various reasons for its importance.
Conducting experiments on real-world systems can be infeasible because experimenting would disrupt the service that is provided by the system. For example, in the case of a mail server, experiments or monitoring could cause delay, or maybe even data loss. Service disruption can sometimes be even dangerous, in addition to being an inconvenience: in the case of a nuclear reactor, delay or loss of data can prove fatal. Timeliness can be as important in such systems as correctness. However, performance analysis and monitoring might be crucial to draw conclusions about maintenance, for example. Another problem with direct experimentation is that the information we are looking for may not be available, or may be complicated to get. For example, in most operating systems, it is difficult to obtain the exact timing of instruction-level events.[8] Also, when conducting experiments on a real-world system, results are often non-reproducible, due to resource dynamics.[1] Another argument on the side of modelling is that it provides the ability of experimenting on different configurations. Investing in a large-scale computer cluster, or the setup of a distributed grid environment is an expensive and tedious process. Investors want to make sure that they

get what they want: they impose performance constraints on the system. This means that they want to know how the system will behave before buying it and setting it up. To predict the behavior, experiments are needed to be conducted. We need to do these experiments on different setups, before finding out which one is the best in the current situation. Changing the hardware or software configuration parameters on a real-world system is very inconvenient - in most cases, it's not doable, because of time and money constraints. Thus, the solution is to simulate the desired system, and run the experiments there. This way, changing the configuration is simple and costless.[8] Another great benefit of simulation is that in a classroom setting, students can learn the principles of high-performance and distributed computing without actual access to a parallel platform.[2]

3.2 Analytical and Simulation Models

The accuracy of a model can vary: we can make an analytical, or qualitative model, in which all definite values are abstracted away - in this case, we get a representation of the system, which can be analysed mathematically to deduce its behavior. When using this method, no experiments can be conducted, we solely rely on theoretical analysis. In contrast, a simulation model is a stochastic model, which is an algorithmic abstraction of the real-world system that can be executed to reproduce the system's behavior. This model is also called a quantitative model, as we can get estimates of the modeled system's quantitative attributes, such as response time or throughput. In other words, we can use a simulation model to conduct performance analysis on a system.[8][9]
When wanting to get a prediction about how a specific system would perform, a theoretical model, in most cases, produces unreliable and unrealistic results - it's not feasible for such accurate predictions. Most research results are obtained via empirical evaluation of experiments.[1] For these reasons, we use the simulation model in this thesis. As we stated before, such a model can be executed, which is called simulation. During simulation, the model behaves like the real system would. Such a model contains more aspects of the real system, compared to the theoretical model, in order to accurately represent the system, while still avoiding unnecessary detail.[8] Creating and executing a simulation model is complicated, computationally expensive and poses a large number of challenges, thus, a good simulation framework can prove of much help when conducting experiments.

4 SimGrid

For the aforementioned reasons, simulation techniques have been historically widely utilised in several areas of computer science, e.g. microprocessor design, network protocol design. Due to this, a lot of effort went into developing the technology and as a result, widely used and reliable simulation frameworks have been developed in these areas. However, there hasn't been a well-developed standard simulation tool for what we talk about in this thesis: execution of distributed applications on distributed computing platforms. Rather, there has only been a number of in-house developed, highly specialized tools to satisfy the need of the community. SimGrid is a more generic simulation framework

that is being developed to be one of the acknowledged and widespread tools for simulation in large-scale distributed computing.[1]
SimGrid's key features include:[1]

- A scalable and extensible simulation engine that implements several validated simulation models, and that makes it possible to simulate arbitrary network topologies, dynamic computational and network resource availabilities, as well as resource failures;
- High-level user interfaces for researchers (who are not necessarily computer science experts, but rather experts on their own field of research) to quickly assemble simulation prototypes in either C or Java;
- APIs for distributed computing developers to create distributed applications that can run seamlessly in either "simulation mode" or "real-world mode", in order to be able to test it on the simulated environment before actually deploying it.

SimGrid is a very active project, both in terms of research and in terms of development. It is a favored tool by researchers, which is proven by the increasing number of papers written where the research was conducted using SimGrid as a scientific instrument. In terms of development, the developer team envisions a number of directions for future work: addition of a model for disk resources; extention of scalability to improve usability in the P2P domain; ability to dispatch simulated nodes over several physical machines.[1] Another important field of research for the SimGrid team is the implementation of the API that has already been mentioned: the Message-Passing Interface (MPI).

5 SMPI

As stated before, MPI is one of the most widely used APIs for communication between nodes in distributed computing. SMPI is a framework for simulating on a single node the execution of parallel applications implemented using the MPI standard. It is part of the SimGrid project and as such, it is built on the SimGrid simulation kernel, benefiting from its fast, scalable and validated network models. SMPI also extends the existing model with other techniques, such as a validated piece-wise linear model for data transfer times between cluster nodes. SMPI simulations also account for network contention - this is done relying on analytical models, rather than packet-level simulation, thus helping the simulation to be fast and scalable.[2]
Three of the main challenges for simulating an MPI application are:

- Accuracy: The prediction of the real-world execution time (the "simulated time") needs to be as accurate as possible, so that reasonable conclusions can be drawn from the experiments.
- Scalability: We want to be able to simulate large-scale applications within a reasonable timescale.
- Speed: It would be advantageous if the simulation time (the actual time of running the simulation) would be as low as possible, compared to the simulated time (the predicted execution time of the real-world application).

Extensive testing was conducted in [2] to verify these qualities of the framework. In these tests, the OpenMPI and MPICH implementations were used to serve as verification benchmarks. The results showed that SMPI predicted the execution time of OpenMPI and MPICH applications for point-to-point, one-to-many and many-to-many applications with an average error value of under 10% in each cases. Using techniques to reduce the memory footprint, SMPI tests were successfully conducted on a scale of up to 448 processors. As a result of the RAM-folding, which was used to reduce the memory footprint of the application, the predicted execution times were underestimates with an average error value of 18.5%, which is higher than in the previous experiments. We have to note here, though, that certain tests weren't successful without the RAM-folding techniques, due to an out-of-memory error. This shows, that although it poses difficulties, reducing the memory usage is vital in SMPI. Also, relying on the CPU sampling approach to reduce the number of computations in an application, SMPI also manages to reduce simulation time by a great margin.
As SMPI is an actively developed project alongside SimGrid, there are a number of research directions. One major development to the project would be a testing framework that would provide the ability to set up experiments across different environments with as little necessary adjustments as possible. Another direction is related to the optimization of the framework's performance, with the utilisation of ideas from the STAR-MPI project.

6 STAR-MPI

Self-Tuned Adaptive Routines for MPI Collective Operations (STAR-MPI) is a set of MPI collective communication routines that are capable of dynamically adapting to system architecture and application workload. The main idea lays in a technique called "delayed finalization of MPI collective communication routines" (DF). For each operation, STAR-MPI maintains a set of communication algorithms. The aim is to postpone the decision of which algorithm to use until after the platform and/or the application are known. This technique bears the potential of platform-specific or application-specific optimalization of an MPI application.[10]
A development idea for SMPI is to apply the same technique there, thus, in a sense, to "implement" STAR-MPI in SMPI. A set of potentially choosable algorithms could be implemented alongside a set of selector mechanisms. With the right mechanisms, the performance of SMPI could greatly improve.

7 References

[1] Casanova, H., Legrand, A. and Quinson, M. (2008), "SimGrid: a Generic Framework for Large-Scale Distributed Experiments", *Proceedings of the Tenth International Conference on Computer Modelling and Simulation*, IEEE Computer Society, Washington, DC, USA, pp. 126-131.

[2] Clauss, P.-N., Stillwell, M., Genaud, S., Suter, F., Casanova, H. and Quinson, M. (2011), "Single Node On-Line Simulation of MPI Applications with SMPI", *International Parallel & Distributed Processing Symposium*, May 2011, Anchorange (AK), United States.

[3] Message Passing Interface Forum (2012), *MPI: A Message-Passing Interface Standard Version 3.0*, HLRS, Stuttgart.

[4] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L. and Woodall, T. S. (2004), "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation", *Proceedings, 11th European PVM/MPI Users' Group Meeting*, September 2004, Budapest, Hungary.

[5] MPICH (2012), *MPICH Overview*, http://www.mpich.org/about/overview/ (accessed 08 April 2013).

[6] Thakur, R., Rabenseifner, R. and Gropp, W. (2005), "Optimization of Collective Communication Operations in MPICH", *Int'l Journal of High Performance Computing Applications*, vol. 19, pp. 49-66.

[7] Buntinas, D., Mercier, D. and Gropp, W. (2007), "Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem", *Parallel Computing*, vol. 33, pp. 634-644.

[8] Jane Hillston (2012), *Performance Modelling*, Lecture 1 ("Modelling and Simulation"), School of Informatics, University of Edinburgh, available at: http://www.inf.ed.ac.uk/teaching/courses/pm/PM-lecture1.pdf (accessed 9 April 2013)

[9] Jane Hillston (2012), *Performance Modelling*, Lecture 13 ("Simulation Models: Introduction and Motivation"), School of Informatics, University of Edinburgh, available at: http://www.inf.ed.ac.uk/teaching/courses/pm/PM-lecture13.pdf (accessed 9 April 2013)

[10] Faraj, A., Yuan, X., Lowenthal, D. K. (2006), "STAR-MPI: Self Tuned Adaptive Routines for MPI Collective Operations", *the 20th ACM International Conference on Supercomputing*, June 2006, Cairns, Australia.