Chapter 1

Evaluation of Communication Libraries For A Parallel Functional Language Implementation

A. D. Al Zain[1], P. W. Trinder[1], G. J. Michaelson[1], H-W. Loidl[2]

Abstract: Special purpose High Performance Computers (HPCs) are expensive and rare resources, but workstation clusters are cheap and becoming common. Emerging GRID technology offers the opportunity to integrate Grid-enabled HPCs into a single HPC. Applications developed in these environments should be based on a language with high-level parallel coordination that hides the complexities of both the Grid infrastructure and the underlying hierarchical, heterogeneous and shared architecture. In this paper we propose the use of GpH (Glasgow parallel Haskell) to be used as a programming language in Grid-enabled systems. The high level coordination in GpH is supported by a sophisticated runtime environment called GUM. GUM is an ideal platform to adapt to the heterogeneous, high latency GRID environment as it supports architecture independence, is readily ported, and is currently supported on both high-latency and low-latency systems. GUM currently uses PVM for program communication infrastructure. Porting GUM to the GRID requires the use of a Globus communication library such as MPICH-G2. This study examines the performance and the behaviour of GUM after it has been ported to Globus.

[1]School of Mathematical and Computer Science, Heriot-Watt University, Edinburgh EH14 4AS, Scotland, Email:{ceeatia,trinder,greg}@macs.hw.ac.uk

[2]Ludwig-Maximilians-Universität München, Institut für Informatik, D 80538 München, Germany, Email: hwloidl@informatik.uni-muenchen.de

1.1 INTRODUCTION

Although computers have become faster and gained more storage, the need is for resources has grown even faster. A first solution is to couple several inexpensive systems to form a more rapid one, as in Beowulf clusters [RBMS97]. However, demand may grow for even more computational power and storage size, where acquiring a new cluster system would be too expensive or an extension of the existing cluster system might be inefficient or introduce a heterogeneous environment. A promising alternative is to couple existing cluster systems. Building clusters out of clusters thus leads to the idea of the computational Grid [FK98].

Computational Grids often involve heterogeneous collections of computers that may reside in different administrative domains, run different software, be subject to different access control policies, and be connected by networks with widely varying performance characteristics [Fos01]. In addition, these heterogeneous collections require varying interconnection speeds to be to be taken into account. Moreover the interconnection, and possibly the computational Grids, are shared, resulting in varying load during the execution of the program.

To characterise a Grid, Foster provides a three point checklist[Fos02]

- A grid system manages resources that are not maintained by centralised control.
- A grid system uses standard, open, general-purpose protocols and interfaces to avoid dealing with an application-specific system.
- A grid system delivers nontrivial qualities of service to its users. Examples are response time, throughput, availability and security.

Grid concepts are realised by software packages like Globus [FK99a] and OGSA [FKNT02].

It is essential that application development in these environments be based on a language with high-level parallel coordination that hides the complexities of both the Grid infrastructure and the underlying hierarchical, heterogeneous and shared architecture. Glasgow parallel Haskell (GpH) [Loi01] currently used on classical high performance computation and abstracts from low level coordination issue such as work and data distribution, and both thread communication and synchronisation [TML02]. The high level coordination in GpH is supported by a sophisticated runtime environment (RTE), GUM [THM+96], see Figure 1.1. GUM, as described by Trinder et al [TML02, TLB+00], is an ideal platform to adapt to the heterogeneous, high latency GRID environment as it supports architecture independence, is readily ported, and is currently supported on both high-latency and low-latency system.

Application support layers that have been deveoped for the Grid system, like LSF (*Load Sharing Facility*) [ZZWD93, Cor01], are primarily designed for high performance distributed execution, with multiple programs cooperating from locations with appropriate resources. In contrast, we focus on the parallel execution

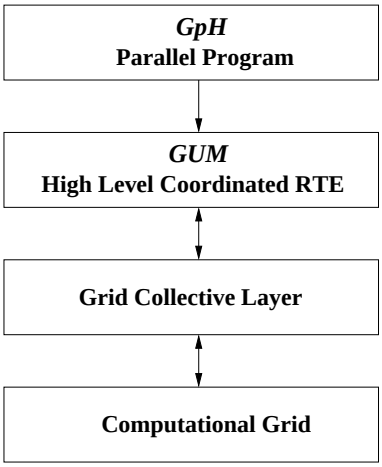

FIGURE 1.1. System Architecture

of a single program which is transparently distributed across a network of high performance processors.

GUM currently uses PVM [PVM93] for program communication infrastructure. Porting GUM to the GRID requires the use of a Globus communication library such as MPICH-G2 [Lab03]. This study examines the performance and the behaviour of GUM after it has been ported to Globus, comparing:

- GUM with PVM, Figure 1.2.a
- GUM with MPI, Figure 1.2.b
- GUM with MPICH-G2/Globus, Figure 1.3.

to identify the impact of both MPI and the Globus infrastructure on GpH programs.

In the following sections, the basic functionality and principles of Globus, GUM, PVM and MPI are discussed. Performance measures for GUM with PVM, MPI, and MPICH-G under Globus are presented for two benchmark GpH programs and their implications for porting GUM are discussed.

1.2 BACKGROUND

1.2.1 Globus Toolkit

The Globus Toolkit is open source software with an open architecture. It is being developed mainly by the Mathematics and Computer Science Division at Argonne National Laboratory with contributions from many developers world-wide. The Globus Toolkit is a collection of software components designed to support the

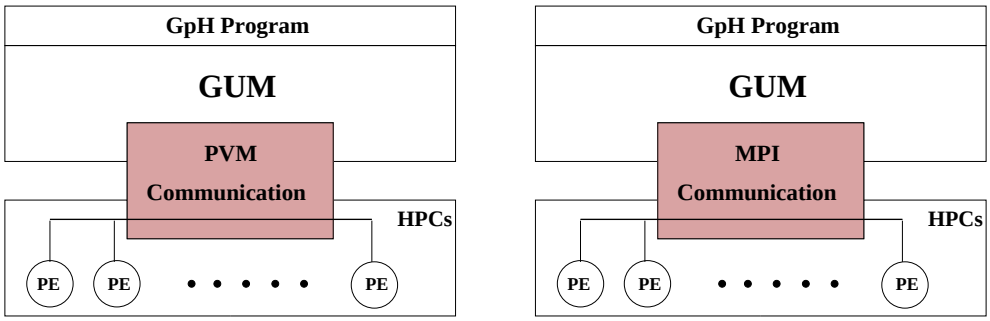

Fig.2.a. GUM-PVM **Fig.2.b.** GUM-MPI

FIGURE 1.2. GUM-PVM & GUM-MPI

development of applications for high performance distributed computing environments or "Grids" [FK99a, FK99b]. The toolkit is based on three main components,

- *Resource Management:* allocation and management of grid resources.
- *Information Services:* providing information about grid resources.
- *Data Management:* accessing and managing data in a grid environment.

1.2.2 GpH & GUM

GpH:

GpH (*Glasgow parallel Haskell*) [THLP98] is a language with very high level coordination, i.e. control of parallel execution. It is a modest and conservative extension of Haskell 98, using the parallel combinator `par` to specify parallel evaluation. The expression `p `par` e`, using Haskell's infix operator notation, has the same value as `e`. Its dynamic effect is to indicate that `p` could be evaluated by a new parallel thread, with the parent thread continuing evaluation of `e`. Also we say that p has been *sparked*, and a thread may subsequently be created to evaluate it if a processor becomes idle [Loi02]. Higher-level coordination is provided using *evaluation strategies*: higher-order polymorphic functions that use `par` and `seq` combinators to introduce and control parallelism [THLP98]

GUM:

GUM (*Graph reduction for a Unified Machine model*) is a highly portable parallel runtime system for GpH, which uses an abstract message passing implementation, originally built around the PVM communication harness. GUM's effectiveness has been demonstrated by parallelising numerous large programs with a relatively small programming effort [LTH+99], achieving wall-clock speedups

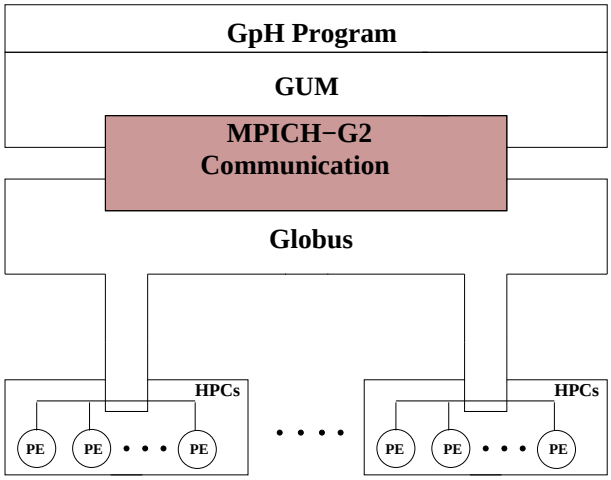

FIGURE 1.3. GUM with Globus

over the equivalent optimised sequential programs. To support GUM on a shared hierarchical heterogeneous architecture like a GRID a new configuration tool has to be developed so GUM can control aspects as work distribution strategy and distributed memory management strategy. Also a dynamic adaptive mechanisms will be augmented to utilise the configuration information. Early version of GUM have already been ported to MPI [PK00] and MPP [Dav96]. The implementation of GUM is based in four main components, we can view them as pillars Figure 1.4:

- *Thread*[1] *Management:* deciding when to generate a new thread and how to schedule the threads.
- *Memory Management:* controlling access to remote data and in GUM it implements a virtual shared heap.
- *Communication:* transferring data and work between PEs.
- *Management* controlling initialisation and termination for the PEs, and load balancing between the PEs

The three GUM implementations which are presented in this paper share the same philosophy of *Thread Management* and *Memory Management*. *Communication* and *Initialisation & Termination* are based on different techniques depending on the GUM's implementation. The remainder of this section describes these differences for GUM's implementation.

[1] A thread is a virtual process

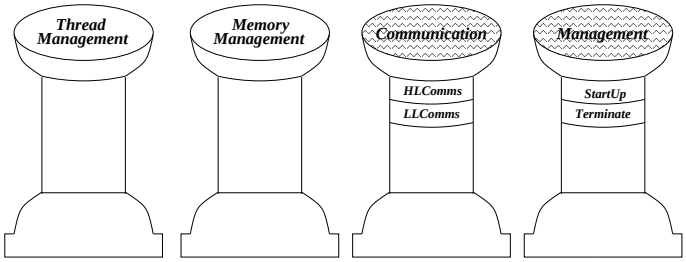

FIGURE 1.4. GUM implementation components

1.2.3 Communication Libraries

PVM

PVM (*Parallel Virtual Machine*) [GBD+94] emerged as one of the most popular cluster message-passing systems in 1992. Although PVM did not originally work on nodes of multicomputers, more recently, multicomputer vendors have offered both layered and native versions of PVM for multicomputer message passing [GKP96].

MPI

The MPI (*Message Passing Interface*) [GLS99] standard defines a library of routines that implement the message passing model. These routines include point-to-point communication functions, where a *send* operation is used to initiate a data transfer between two concurrently executing program components, and a matching *receive* operation which is used to extract that data from system data structures into application memory space. It also provides collective operations such as broadcast and reduction that explicitly involve multiple processors.

MPICH

MPICH [GLDS96] is a popular implementation of the MPI standard. It is a high-performace, highly portable library originally developed as a collaborative effort between Argonne National Laboratory and Mississippi State University. The first version of MPICH was developed in parallel with the MPI-1 standard, to demonstrate that the standard was not becoming too complex and could be implemented quickly.

The portability of MPICH originates from its design. There are two layers. The major part of the code is implemented device independently on top of the Abstract Device Interface (ADI). This makes it possible to easily port MPICH to new hardware architectures. The minor device dependant part implements the ADI. It should preferably be implemented by the hardware vendor for a maximum of efficiency. Each such implementation is called an MPICH device.

MPICH-G: A Grid enabled MPI

The Globus Toolkit enables access to various computational resources, but it cannot itself provide a convenient way to use several resources simultaneously. A Globus-enabled version of MPICH can solve this by providing a Globus-device. The first version, called MPICH-G is based on the Nexus Communication Library [FKT96].

MPICH-G2, the current implementation of MPICH-G, no makes longer use of the Nexus [FKT96] library. It is said to have re-implemented the "good" parts of Nexus and improved the others [AHS+02]. MPICH-G2 is supported by the Globus Toolkit since version 1.1.4 and conforms to the MPI standard 1.1 with some additional features of MPI 2.0.

1.3 MEASUREMENTS

We have evaluated the three GUM versions using the GpH benchmark programs `parFib` and `maze` described below. All the measurements use up to 30 nodes of a Beowulf cluster, consisting of Linux RedHat 8.0 workstations, each with a 533MHz Celeron processor, 128Kb cache, 128MB of DRAM and 5.7GB of IDE disk. The workstations are connected with a 100Mb/s fast Ethernet switch.

1.3.1 `parFib`

`parFib` [Loi01] computes the number of calls to the Fibonacci function. It uses arithmetic heavily with a double recursive structure. The granularity in `parFib` can be controlled by specifying the threshold for parallel invocation which will help manage the computation size. In this experiment we calculate `parFib 40`.

1.3.2 `maze`

`maze ??` traverses a tree with exactly one path from the entrance to the exit (one leaf), as shown in Figure 1.5. In this experiment, `maze` is a tree with a branching factor of 11 for the root and 10 for all other levels. The depth of the root's subtree is 10, but the subtree containing the exit has a depth of 9.

1.3.3 Performance Results:

Tables 1.1 & 1.2, and Figure 1.6 show the runtime (Seconds) for the three versions of GUM. Tables 1.3 and 1.4 show differences in times between PVM and MPI, PVM and MPICH, and MPI and MPICH, as proportions of the time for the first in each pair, for parfib and `maze` respectively,

Overall, the differences are small. However, for `parFib`, MPI offers better times and MPICH under Globus worse times, than PVM. These trends become more marked as the number of processors increases. In contrast, for `maze`, there is little difference between the PVM and MPI times, with MPI better for small numbers of processors. Up to 16 processors, MPICH under Globus is somewhat

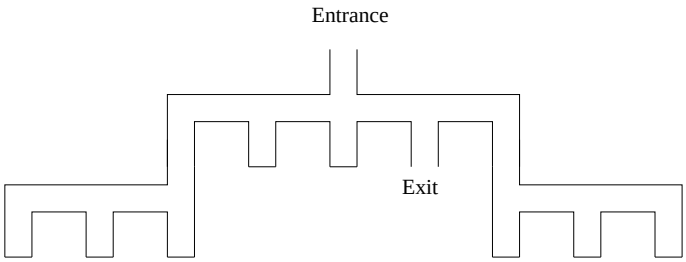

FIGURE 1.5. A maze as a tree

Comm. Libraries	Measures	PEs No. 1	2	4	8	16	30
PVM	*Total*	409.90	213.12	98.27	40.01	23.69	11.69
MPI	*Total*	406.23	210.78	94.24	39.69	20.87	9.92
MPICH-G2	*Total*	414.66	220.31	104.56	58.63	27.03	16.01

TABLE 1.1. `parFib` **Runtimes in Seconds**

worse than both, but slightly better for 30 processors. However, the differences here are less marked than for `parFib`.

The comparison of MPI and MPICH under Globus shows that MPI is consistently faster for `parFib` and marginally faster, except at 30 processors, for `maze`. Assuming considerable consistency betweem MPI and core MPICH, overall these results suggest that the use of Globus has more impact than that of MPI on GUM behaviour.

1.4 FUTURE WORK

We have shown that implementing GUM with MPICH has little impact on the behaviour of GpH programs on a sole use Beowulf cluster. However there might be more impact if multiple or shared clusters were used. The GUM load balancing mechanism has been designed to work in a flat architecture where all the PEs share one domain. In contrast, in the Grid architecture there is a complex infrastructure imposed on a hierarchical, heterogeneous and shared architecture.

Comm. Libraries	Measures	PEs No. 1	2	4	8	16	30
PVM	*Total*	946.83	308.44	180.19	102.30	51.58	50.78
MPI	*Total*	932.25	301.72	175.58	104.01	52.01	51.80
MPICH-G2	*Total*	950.81	313.60	184.18	106.78	55.10	48.98

TABLE 1.2. `maze` **Runtimes in Seconds**

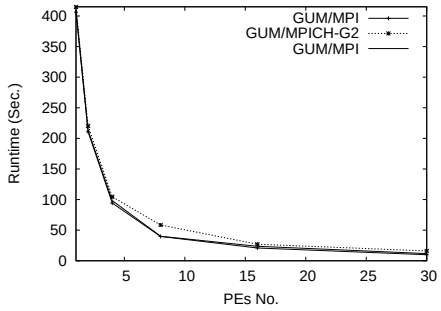

Fig.1.6.a. `parFib`

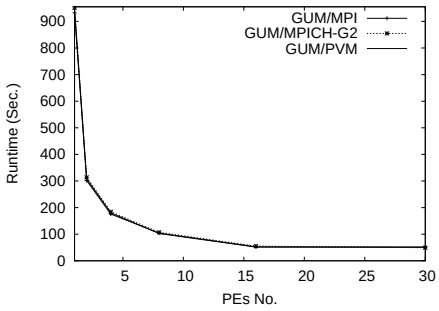

Fig.1.6.b. `Maze`

FIGURE 1.6. GUM runtimes

Comm. Libraries	PEs No.					
	1	2	4	8	16	30
$\frac{PVM-MPI}{PVM}$	0.00	0.01	0.04	0.00	0.12	0.15
$\frac{PVM-MPICH}{PVM}$	-0.01	-0.03	-0.06	-0.46	-0.14	-0.36
$\frac{MPI-MPICH}{MPI}$	-0.02	-0.04	-0.11	-0.48	-0.29	-0.61

TABLE 1.3. `parFib`

We next plan to explore how to run GUM effectively on Grid compliant clusters using Globus with MPICH as the communications and load distribution framework. We intend to augment GUM with mechanisms to monitor work distribution, network behaviour and memory use, to enable adaptive dynamic thread placement to optimise processing.

REFERENCES

[AHS+02] R. J. Allan, D. Hanlon, G. Smith, R. F. Fowler, and C. Greenough. A Globus Developers' Guide with Installation and Maintenance Hints. WWW Page, July 2002. <URL:`http://esc.dl.ac.uk/StarterKit/PS/globus_guide.ps`>.

[Cor01] Platform Computing Corporation. LSF Reference Guide. WWW Page, June 2001.

Comm. Libraries	PEs No.					
	1	2	4	8	16	30
$\frac{PVM-MPI}{PVM}$	0.01	0.02	0.02	-0.01	-0.01	-0.02
$\frac{PVM-MPICH}{PVM}$	0.00	-0.01	-0.02	-0.04	-0.07	0.03
$\frac{MPI-MPICH}{MPI}$	-0.02	-0.04	-0.04	-0.02	-0.05	0.05

TABLE 1.4. `maze`

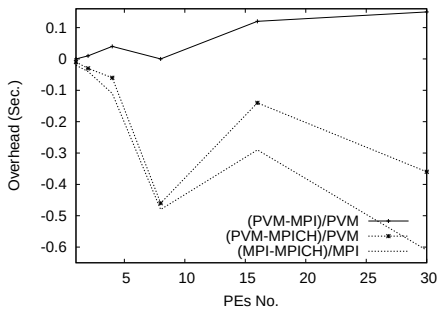

Fig.1.7.a. `parFib`

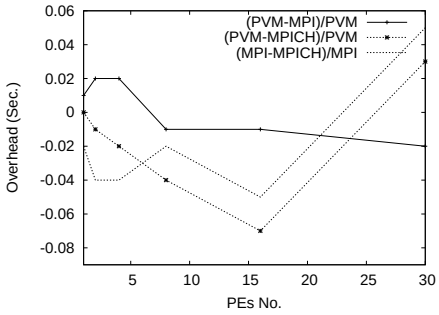

Fig.1.7.b. `Maze`

FIGURE 1.7. GUM Overhead

<URL:`http://batch.web.cern.ch/batch/doc/ref_4.2.pdf`>.

[Dav96] Kei Davis. MPP Parallel Haskell. In *Draft Proceedings IFL'96 — Intl. Workshop on the Implementation of Functional Languages*, LNCS 1268, pages 49–54, Bonn/Bad-Godesberg, Germany, September 1996. Springer.

[FK98] Ian Foster and Carl Kesselman. Computational Grids. *The Grid: Blueprint for a Future Computing Infrastructure*, 1998.

[FK99a] Ian Foster and Carl Kesselman. The Globus project: a status report. *Future Generation Computer Systems*, 15(5–6):607–621, 1999.

[FK99b] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kufmann, 1999.

[FKNT02] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid. an open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002. <URL:`http::://www.globus.org/research/papers/ogsa.pdf`>.

[FKT96] Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37(1):70–82, 1996.

[Fos01] Ian Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–??, 2001.

[Fos02] Ian Foster. What is the Grid? A Three Point Checklist. *Daily news and information for the global grid community*, 1(6), July 22 2002. <URL:`http://www.gridtoday.com/02/0722/100136.html`>.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongerra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine*. MIT, 1994.

[GKP96] G. A. Geist, J.A. Kohl, and P. M. Papadopoulos. PVM and MPI: A comparison of features. *Calculateurs Parallels*, 8(2), 1996.

[GLDS96] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI Message-Passing Interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[GLS99] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT, second edition, 1999.

[Lab03] High-Performance Computing Laboratory. MPICH-G2. WWW page, January 2003. <URL:`http://http://www.hpclab.niu.edu/mpi/`>.

[Loi01] H-W. Loidl. Glasgow Parallel Haskell. WWW page, January 2001. <URL:`http://www.cee.hw.ac.uk/~dsg/gph/`>.

[Loi02] H-W. Loidl. Load Balancing in a Parallel Graph Reducer. In Kevin Hammond and Sharon Curtis, editors, *Trends in Functional Programming*, volume 3, pages 63–74, Bristol, UK, 2002. Intellect.

[LTH+99] H-W. Loidl, P.W. Trinder, K. Hammond, S.B. Junaidu, R.G. Morgan, and S.L. Peyton Jones. Engineering Parallel Symbolic Programs in GPH. *Concurrency — Practice and Experience*, 11:701–752, 1999.

[PK00] Mari Plümacher and Ulrike Klusik. MPI-Port of GUM. WWW page, July 2000. <URL:`http://www.mathematik.uni-marburg.de/~klusik/mpigum/`>.

[PVM93] *Parallel Virtual Machine Reference Manual*. University of Tennessee, Aug 1993.

[RBMS97] D. Ridge, D. Becker, P. Merkey, and T. Sterling. Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs. In *IEEE Aerospace Conference*, pages 79–91, 1997.

[THLP98] P.W. Trinder, K. Hammond, H-W. Loidl, and S.L. Peyton Jones. Algorithm + Strategy = Parallelism. *Journal of Functional Programming*, 8(1):23–60, January 1998.

[THM+96] P.W. Trinder, K. Hammond, J.S. Mattson Jr., A.S. Partridge, and S.L. Peyton Jones. GUM: a Portable Parallel Implementation of Haskell. In *PLDI'96 — Conf. on Programming Language Design and Implementation*, pages 79–88, Philadephia, USA, May 1996.

[TLB+00] P.W. Trinder, H-W. Loidl, E. Barry Jr., K. Hammond, U. Klusik, S.L. Peyton Jones, and Á.J. Rebón Portillo. The Multi-Architecture Performance of the Parallel Functional Language GPH. In A. Bode, T. Ludwig, and R. Wismüller, editors, *Euro-Par 2000 — Parallel Processing*, volume 1900 of *LNCS*, pages 739–743, Munich, Germany, 29.8.-1.9., 2000. Springer-Verlag.

[TML02] Phil Trinder, Greg Michaelson, and Hans Wolfgang Loidl. Adaptive execution for high-performance grid computing. Technical report, Dept of Computer Science, Heriot-Watt University, September 2002.

[ZZWD93] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software - Practise and Experience*, 23(12):1305–1336, 1993.