Grid Computing in an Academic Environment

George C. Wells and Gregory A. Atkinson
Department of Computer Science, Rhodes University
Grahamstown, 6140, South Africa
G.Wells@ru.ac.za

Abstract *This paper describes an investigation into the provision of grid computing facilities in a typical university environment. In particular, it focuses on the use of the Java programming language and the ProActive grid framework. The experience of deploying ProActive in a heterogeneous environment (Windows and Linux) is discussed. The results of the implementation of a bioinformatics application using ProActive are presented and compared with the results of a similar application using a Linda-based approach.*

Keywords: Grid Computing, Java, ProActive, Bioinformatics

1 Introduction

Modern science is becoming increasingly dependent on data reduction and modeling, and manipulation of very large data sets—complex problems requiring large amounts of computational time. Grid computing provides a powerful tool that coordinates the resources required to solve scientific problems in a cost-effective manner, while encouraging the use of distributed resources and collaboration.

The term "grid computing" is interpreted differently in various contexts. For our purposes we adopt the following definition, proposed in [1]:

> *The term "the Grid" refers to a network-based computing infrastructure providing security, resource access, information and other non-trivial services that enable the controlled and coordinated sharing of resources among "virtual organisations" formed dynamically by individuals and institutions with common interests.*

We distinguish between a grid and a *cluster*, in that the latter term is used for tightly-coupled, dedicated systems.

This research project involved a detailed investigation of frameworks and methods for developing computing grids using networks of commodity computers, such as those found in most academic institutions. Similar investigations have been performed in [2, 3]. Due to the wide usage of the Java programming language in universities we focused on grid computing frameworks that supported Java, and specifically on the ProActive framework[4], developed by the Active Objects, Semantics, Internet and Security (OASIS) project team as a research initiative of the French National Institute for Research in Computer Science and Control (INRIA). It inter-operates with and builds on several standards, such as HTTP, JINI, OGSi[1], RMI, Sun Grid Engine and Globus. The project has also had contributions from a number of external developers, enabled by the use of the GNU Lesser General Public License (LGPL).

The next section of this paper gives further details of the ProActive framework and our experience of deploying it in a typical university setting. This is followed by a description of a bioinformatics application and its implementation using ProActive. The performance results from testing this application are then presented, and compared with prior results obtained using a Linda system. The final section of the paper presents some conclusions and discusses areas for further investigation.

2 Overview of ProActive

ProActive is a middleware solution for grid computing. The ProActive manual[5] defines ProActive as "a *Grid* Java library for *parallel*, *distributed* and *concurrent* computing, also featuring *mobility* and *security* in a uniform framework", and asserts that it "provides a comprehensive API allowing [the simplification of programming applications] that are distributed on *Local Area Networks* (LAN), on [a] *cluster of workstations*, *P2P desktop Grids*, or on *Internet Grids*".

ProActive can be deployed using either a clus-

[1] Open Grid Services Infrastructure.

ter, where it is explicitly started on each cluster node and the nodes are explicitly aware of the other nodes in the cluster, or in a peer-to-peer (P2P) configuration. The P2P infrastructure is built upon the cluster mode infrastructure, but with extensions to facilitate a significantly more dynamic grid. The purpose of the P2P infrastructure is for maximum utilisation of an institution's spare CPU cycles[5]. Organisations may not have a dedicated cluster, but most universities and research centres have a large number of desktop computers that are very seldom utilised for more than 60% of the day.

Peer nodes can dynamically join and leave the grid, and are managed by the ProActive P2P infrastructure's *P2PService*. The P2P infrastructure is overlaid on a dynamic network of Java Virtual Machines, where each peer executes the P2PService and acts as a computational node. The P2PService comprises a few *Active Objects*[5], that facilitate the integration and management of the peers. Active Objects are based on a concept called *Transparent Remote Objects* (TROs)[6], which builds extensively upon Java's Remote Method Invocation (RMI) framework[7]. This approach allows a single application to be deployed over multiple distributed resources and is itself based on the Java// (pronounced *Java parallel*) framework, which aims to provide "seamless sequential, multi-threaded and distributed programming"[6].

A method call on an Active Object returns immediately and a *Future* object is returned in the place of the result. When the method terminates, the Active Object updates the *Future* it returned with the actual result[8]. The calling thread continues execution and only blocks if (1) it requires the returned object while it is still a Future, or (2) the returned object is a primitive. This provides *automatic continuation*[5] and wait-by-necessity, asynchronous communication. Explicit synchronisation can also be performed using the ProActive API, if necessary.

ProActive also provides an exceptionally useful utility called *Interactive Control and Debugging of Distribution* (IC2D), which gives the user of the Grid complete control and monitoring capabilities over all deployed applications. An application can be launched from within the IC2D interface or it can be attached to an application that is already executing. IC2D works by installing a *SpyListener* Active Object on each node. The SpyListeners intercept all messages passed to and from their respective nodes and report this information, along with the state of each Active Object, back to IC2D. The user can then monitor and control the application in real-time during its execution, with both graphical and textual interfaces. IC2D also provides for dynamic change of deployment and drag-and-drop migration of executing tasks. This is an extremely useful and impressive feature of IC2D: using the graphical interface, Active Objects can be dragged from one node and dropped onto another, even while they are executing, without any effect on the running application. This provides a simple, manual mechanism for load balancing.

ProActive also provides fault tolerance, security (based on the Java Cryptography Extension, JCE), file transfer mechanisms (using any one of several different protocols), and support for web services[5].

2.1 Experience with ProActive Deployment

The deployment strategy was to first develop a testbed of virtual machines, that are easy to create and recreate, to experimentally find a working solution for deploying a ProActive grid. Once a working solution had been developed the next phase was to deploy in a laboratory of just over 90 workstations (identical Intel Pentium machines, running both Fedora Core 4 Linux and Microsoft Windows XP).

To best meet the requirements of the Grid and best utilise the resources available, a number of design decisions were made, based on the needs of the academic environment. Firstly, because the workstations could arbitrarily be running in either Linux or Windows at any time, both operating systems needed to be configured as grid nodes as similarly and as simply as possible (this was one of the primary reasons for selecting ProActive—its cross-platform portability). The highly dynamic nature of these workstations led to the first design decision—using the peer-to-peer infrastructure of ProActive. The P2P infrastructure is completely decentralised and consequently the grid would not have to rely on any one node to function. It would also allow for the workstations to join and leave the grid as users made use of them or as they were restarted into one of their two operating systems.

The second design decision was to disable and enable each grid node on each workstation as users logged in and out respectively. This policy would maximise the use of free CPU cycles, without hindering the productivity of any of the users of the workstations. This idea, simple as it may seem, proved to be significantly more difficult to implement than originally anticipated (see below).

Central storage of ProActive and its configura-

tion was considered a necessity and therefore the third design decision. It would be incredibly tedious, if not almost impossible, to manually alter each of ninety computers every time a change needed to be made to the grid.

The ProActive P2P infrastructure is started as a daemon service by a script provided with ProActive. Using a simple telnet client, one can connect to the P2P service and issue commands to *start*, *restart* or *stop* the service, or to *flush* the log file. This script also hides the complexity of the command line options and environment settings required when starting a ProActive application. To extend this functionality, a new Perl script was developed to provide the same set of commands, but from the command line, and to provide additional functionality (e.g. hostname and IP address checking and execution priority selection).

The Fedora Core 4 Linux image on the workstations was configured with an unprivileged user, which provided a secure context for ProActive to execute within. ProActive was integrated with the Gnome Display Manager (GDM) that facilitates user logins, and which has a number of customisable scripts. These scripts were customised as follows: on *initialisation* (i.e. when the GDM becomes visible, before any user is logged in) ProActive is synchronised from the central location, allowing any configuration updates, and started; on *post-login* (i.e. when a user has logged in) ProActive is stopped. In this way, each node executes in a secure context and joins or leaves the grid as users make use of the workstations.

The Windows XP installation on each of the workstations belonged to a domain, using Microsoft's Active Directory. All user accounts were registered on the domain and their home directories were centrally stored on a shared file system. Therefore, it was simple to create a domain user with a home directory to securely encapsulate ProActive. A Windows service, *GridService*, was developed in C# to control the starting and stopping of the P2P infrastructure. GridService executed ProActive, via the Perl script, in the context of another, local system user, further encapsulating ProActive into a secure, read-only execution environment. When the workstation booted, GridService would start the P2P infrastructure and run it with low priority at all times, as it was not possible to configure the Windows XP environment to only execute ProActive when users were logged out. Thus ProActive was made to function on the Windows platform, but not to the full extent intended. This was no fault of ProActive's, which functioned without fault, it was simply due to the inflexibility of the Microsoft Windows platform and the lack of true multi-user functionality.

Three strategies were attempted to circumvent this limitation under Windows. Firstly, an *Event Handler Dynamically Linked Library (DLL)* was developed for integration with the *Winlogon* subsystem. This DLL implemented an interface that defined several functions which were called on system startup and shutdown; user login and logout; desktop lock and unlock; and screensaver start and stop. The DLL functioned properly and was able to correctly start ProActive on system startup or stop it on system shutdown or user login, but as soon as it started ProActive on user logout, the processes were terminated by the desktop closing. Secondly, a similar configuration was attempted using domain group policy, and thirdly a Windows service was developed using C#, but in each case the outcome was the same—the logout procedure terminated the grid processes.

Of course a solution to the problem does exist, because a number of applications and even Java applications have the ability to run as Windows Services, but the implementation became increasingly convoluted and complex as each new possibility was explored. Finally, a possible solution was found: *Java Service Wrapper*[9], a free, open-source tool for creating Windows Services from Java applications. It recognises and claims to have overcome the problem of process termination when desktops close, and would be a likely solution. Unfortunately, time did not permit any further investigation of this approach.

3 The Bioinformatics Application

This application was designed to benchmark ProActive and compare it with a Java implementation of the Linda[2] coordination language (TSpaces from IBM[10]). A *Grid Parallel Motif Searching* application, *GridPMS*, was designed to be as similar as possible to the original TSpaces implementation of the *Parallel Motif Searching* (*PMS*) application developed by Tim Akhurst[11, 12]. Similar work has been reported in [13, 14]. The primary aim of our experiment was to explore the feasibility of using a high-level, abstract framework such as ProActive by investigating its advantages and disadvantages for grid computing in a typical academic environment.

[2]Linda is a registered trademark of Scientific Computing Associates.

Th GridPMS application utilises the classic replicated-worker (or master-worker) pattern to search DNA sequences for subsequences corresponding to specific proteins (or *motifs*). The motifs are expressed as regular expressions, which are then used with the Java `java.util.regex` package in order to locate the motifs within DNA sequences. More details of the bioinformatics field and the molecular biology underlying this problem domain may be found in [12]. The dataset used by both versions of the application comprised one quarter of the human genome, a 1GB text file. The Grid and TSpaces nodes used a similar set of regular expressions, on slightly overlapping segments of the DNA sequence residing in each node's memory.

The biggest difference between the TSpaces and Grid applications was the method by which the DNA sequence was divided and distributed. In the original PMS application the complete sequence was shared to each node via a network filesystem, and each node received instructions regarding which segment of the sequence to read into memory and process. GridPMS was constructed in a client-server fashion, where the master process, with access to the dataset, spawned the workers and passed to them the complete regular expression list together with their segment of the DNA sequence.

4 Testing and Results

The computers used as Grid nodes were an homogeneous collection of Intel 3.0GHz Hyper-threading Pentium 4's with 1GB memory interconnected by a 100Mbps switched Ethernet local area network, each running Java 1.5.0 on Fedora Core 4 Linux and Microsoft Windows XP. This differed slightly from the Intel 2.4GHz Pentium 4's with 512MB memory running Java 1.4.2 on Red Hat Linux 3.1.10 interconnected by the same network, used in [12]. Although this is a significant difference, compounded by using multiple machines, the performance results may be compared by normalising the measured experimental times.

4.1 Results

The performance increase achieved by adding nodes to the PMS is represented in Figure 1, which is a plot of the speedup achieved. The execution time for each test run was measured for the entire duration of each run, including the time taken to distribute the dataset over the network and receive the results. The execution time when 5 nodes were used was of the order of 2.5 hours, but in each test case approximately 5 minutes was spent distributing the dataset over the network, an effect that became more apparent as more nodes were added and the execution time shortened.

Due to the complexity of the application and the large dataset, it was discovered that no less than 5 nodes could be used before the master process exhausted its memory—a point in favour of distribution over the grid. An option would have been to compress the dataset, as the DNA data is essentially 2-bit data (representing the four DNA *bases*) and it is quite wasteful transferring it as 8-bit or 16-bit text.

Before the results could be normalised, the execution time for only one node needed to be calculated. This was achieved by fitting the data using non-linear least-squares regression, giving Equation 1. The trend in Figure 1 appeared to be an exponential increase tending to a constant value, or simply a constant minus an exponential, which is exactly what Equation 1 represents (the statistical R-squared value, a measure of goodness of fit, was calculated to be 99.87%, indicating a near perfect fit). This equation allowed $s'(n)$, the normalised speedup, to be calculated and hence compared to the results achieved in [12].

$$s'(n) = -\alpha^{\beta n - \gamma} + \frac{1}{s(1)}\alpha^{5\beta-\gamma} + 1 \qquad (1)$$

where n = number of nodes; s = speedup from n nodes; s' = normalised speedup from n nodes; α, β and γ are parameters solved for by the non-linear least-squares regression fit.

A byproduct of the curve-fitting to Equation 1 is the interpretation of the constant $\frac{1}{s(1)}\alpha^{5\beta-\gamma} + 1$, which quantifies the maximum theoretical speedup achievable with an infinite number of nodes. The theoretical maximum was calculated to be 69.3 times (the maximum speedup measured was 50.5 times, using 90 nodes).

Figure 1 shows that the application continues to benefit from speedup through adding more processing nodes to the problem up to the limit tested (90 computers). However, Figure 2 suggests that once past a certain point very little additional benefit is derived by adding more nodes. The optimal number of nodes for an application is very much a qualitative measure and depends entirely upon the application, its parameters and its computation to communication ratio. It can be judged from Figure 2 that using more than 50 nodes produces little benefit in the case of GridPMS under our test conditions.

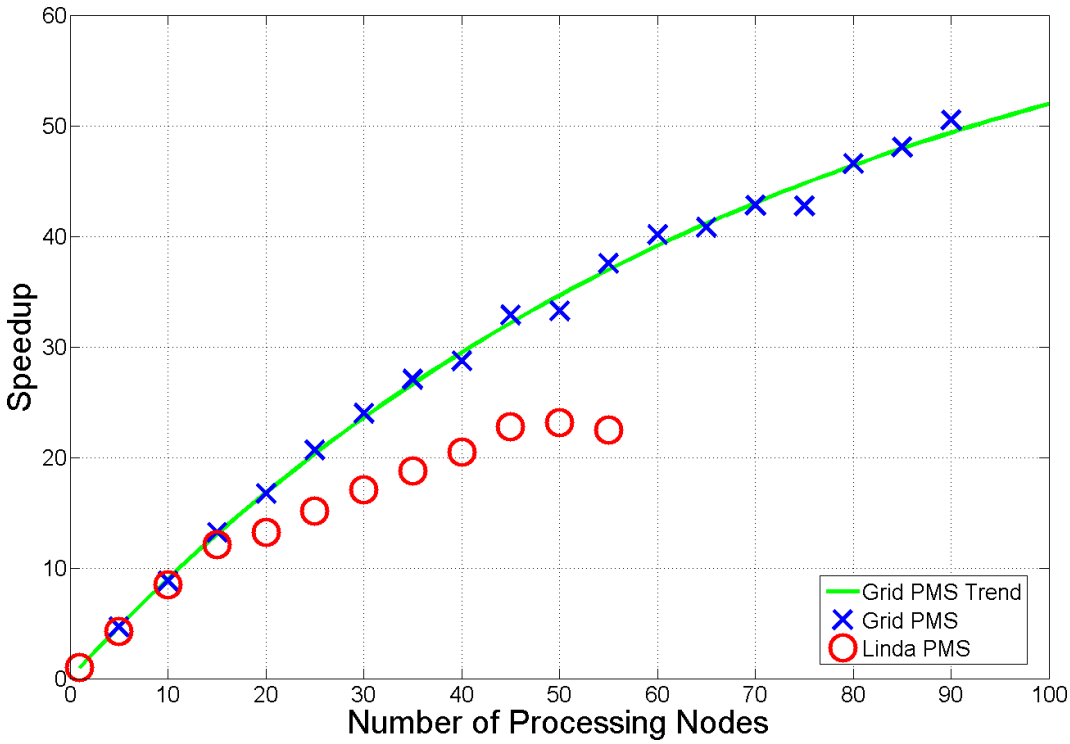

Figure 1: Speedup achieved by GridPMS and the Linda-based PMS

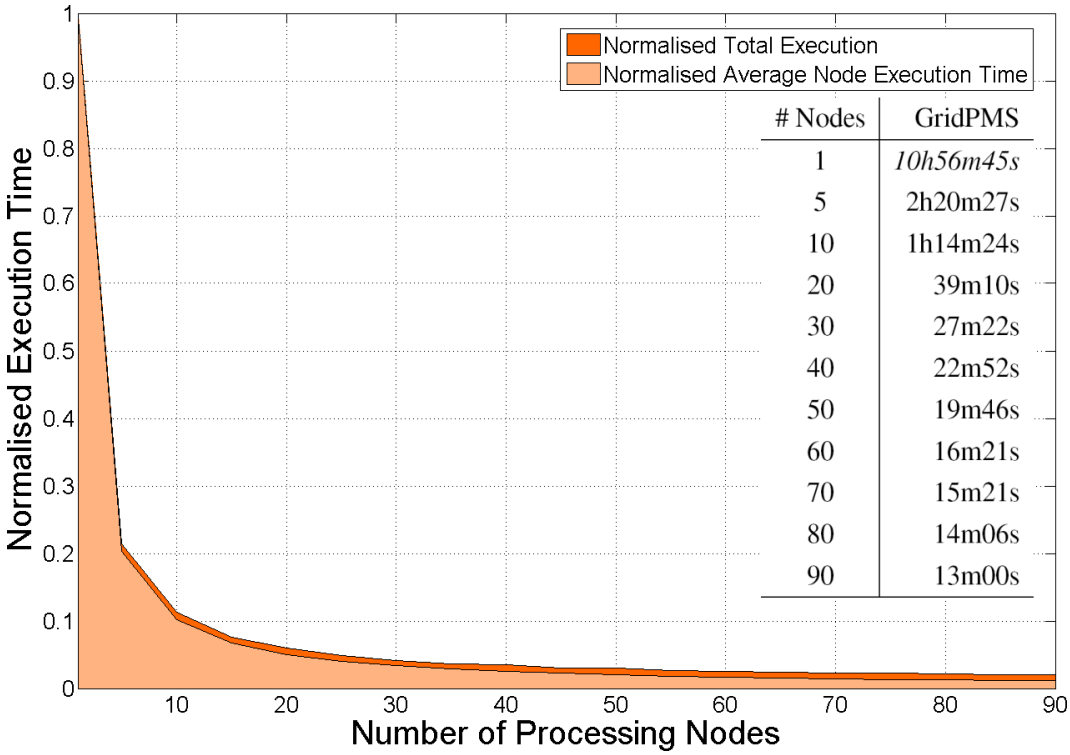

Figure 2: Normalised execution time achieved by GridPMS

The efficiency of a distributed application executing in parallel, calculated as $e = s'(n)/n$, quantifies how well the application utilises the distributed resources it has at its disposal. This performance measure shows the most significant difference between the Grid and Linda implementations of the PMS application. GridPMS is between 10% and 30% more efficient than the Linda implementation in their use of between 5 and 45 computational nodes, evident in Figure 3. The difference in efficiency may be due to the difference in the data distribution approach, as discussed previously, rather than any fundamental differences between ProActive and the Linda coordination language.

5 Conclusions

One major hurdle encountered during this research and perhaps a drawback to the wide-spread use of ProActive was its documentation and the availability of information and support. The ProActive Manual[5] was used as far as possible, but unfortunately the standard of the documentation was very low and no supplementary online resources existed. For example, the grammar in parts was so poor that it made those sections incomprehensible, and certain sections referenced by the text, often source code listings, were absent. On many occasions it became necessary to resort to reading the actual Java source code to understand its functionality and requirements. At times this was frustrating, but it is our only significant criticism of ProActive.

Bar this one criticism, developing applications using the ProActive API was a pleasure. ProActive's Active Object pattern, programming model and API made for a simple approach to solution visualisation and application development. The complete ProActive library is neatly packaged into Java Archives (JARs) and therefore easily linked into an application. ProActive provides instructions[5] for using its API within the Eclipse Integrated Development Environment, providing an excellent environment for developing ProActive applications.

The GridPMS bioinformatics application has demonstrated that both the Grid and the application can scale up to 90 computers (the maximum available to us) without a degradation in performance. The results achieved were better than those achieved previously when utilising the Linda coordination language for a similar application. The application required little modification to operate over the Grid and considering the complex services provided by ProActive for monitoring, administering and maintaining Grid applications, the results are very pleasing. These findings suggest that ProActive is an ideal grid framework for many academic and research centres, allowing networks of commodity workstations to be fully utilized.

5.1 Future Work

ProActive provides greater functionality than was utilised in this study, and a number of areas were identified where further research could be conducted. Of particular interest is developing Web Services that provide an interface to the ProActive Grid. Active Objects can be exported as a Web Service[5, 8] and deployed onto a Jakarta Tomcat web server. This mechanism could make the Grid significantly more usable by providing the ability to create client applications in any Web Service enabled programming language.

The Globus Toolkit is a widely-used infrastructure for grid computing[15], and would form the basis of an excellent comparison to this research. A Globus-based grid could be deployed and a similar application could be developed for testing and benchmarking against that presented in this paper. Furthermore, the Commodity Grid (CoG) Kits, that provide an interface for the Java[16], Python and Perl programming languages to the C/C++ based Globus infrastructure, could be investigated.

Acknowledgments

This work was supported by the Distributed Multimedia Centre of Excellence at Rhodes University, funded by Telkom, Business Connexion, Comverse, Verso Technologies, StorTech, Tellabs, Amatole Telecommunication Services, Mars Technologies, Bright Ideas Projects 39 and THRIP.

References

[1] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[2] V.K. Saini and Q. Duan. Building an experimental Globus-based grid computing system in an university environment. *J. Comput. Small Coll.*, 21(5):188–188, 2006.

[3] M.A. Holliday, B. Wilkinson, and J. Ruff. Using an end-to-end demonstration in an undergraduate grid computing course. In *ACM-SE*

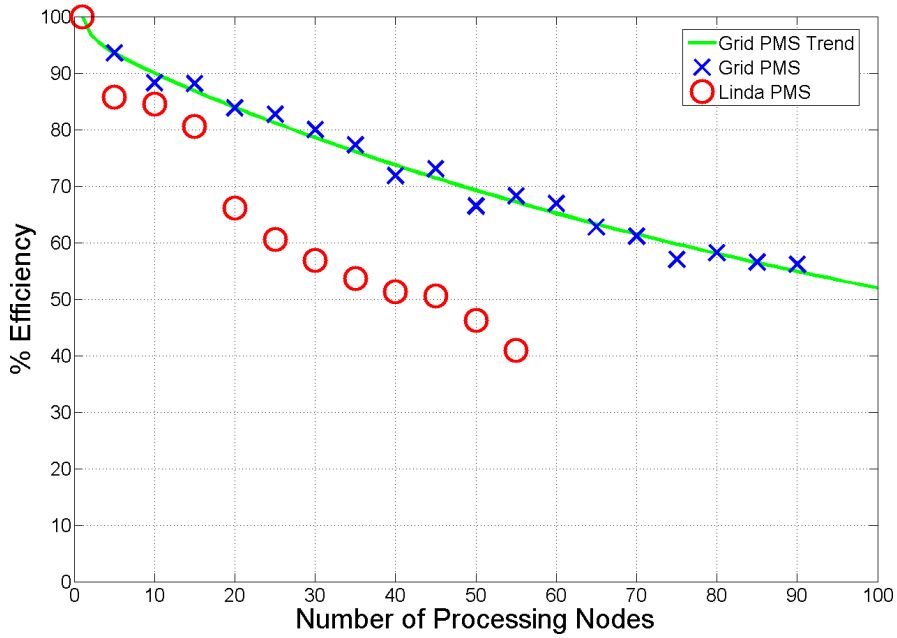

Figure 3: Efficiency achieved by GridPMS and the Linda-based PMS

44: Proc. 44th annual Southeast regional conference, pages 702–707, New York, NY, USA, 2006. ACM Press.

[4] INRIA. ProActive. URL: `http://-www-sop.inria.fr/oasis/ProActive/`.

[5] OASIS Research Team. *ProActive: A Comprehensive Solution for Grid Computing*, v3.1 edition, April 2006.

[6] D. Caromel, W. Klauser, and J. Vayssiere. Towards seamless computing and metacomputing in Java. *Concurrency: Practice and Experience*, 10(11–13):1043–1061, September–November 1998.

[7] A. Wollrath, R. Riggs, and J. Waldo. A distributed object model for the Java system. In *Second USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 219–232, June 1996.

[8] OASIS Research Team. *ProActive Reference Booklet*, v3.0 edition, November 2005.

[9] Tanuki Software. Java Service Wrapper. URL: `http://wrapper.tanukisoftware.org/`.

[10] P. Wyckoff, S.W. McLaughry, T.J. Lehman, and D.A. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[11] T.J. Akhurst. The role of parallel computing in bioinformatics. Master's thesis, Rhodes University, 2004.

[12] G.C. Wells and T.J. Akhurst. Using Java and Linda for parallel processing in bioinformatics for simplicity, power and portability. In *Proc. International Conference on Advances in the Internet, Processing, Systems, and Interdisciplinary Research (IPS-USA-2005)*, Cambridge, MA, USA, June 2005.

[13] M.S. Sousa and A.C.M.A. Melo. PackageBLAST: an adaptive multi-policy grid service for biological sequence comparison. In *SAC '06: Proc. 2006 ACM Symposium on Applied Computing*, pages 156–160, New York, NY, USA, 2006. ACM Press.

[14] M.K. Gardner, W. Feng, J.S. Archuleta, H. Lin, and X. Ma. Grid applications—parallel genomic sequence-searching on an ad-hoc grid. In *SC '06: Proc. 2006 ACM/IEEE conference on Supercomputing*, page 104, New York, NY, USA, 2006. ACM Press.

[15] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[16] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(8–9):643–662, 2001.