On the Origin of Grid Species: The Living Application

Derek Groen[1,2], Stefan Harfst[1,2], and Simon Portegies Zwart[1,2,3]

[1] Section Computational Science, University of Amsterdam,
Amsterdam, The Netherlands
djgroen@uva.nl
[2] Astronomical Institute "Anton Pannekoek", University of Amsterdam,
Amsterdam, the Netherlands
[3] Sterrewacht Leiden, Universiteit Leiden
Leiden, The Netherlands

Abstract. We present the living application, a method to autonomously manage applications on the grid. During its execution on the grid, the living application makes choices on the resources to use in order to complete its tasks. These choices can be based on the internal state, or on autonomously acquired knowledge from external sensors. By giving limited user capabilities to a living application, the living application is able to port itself from one resource topology to another. The application performs these actions at run-time without depending on users or external workflow tools. We have applied this new concept in a special case of a living application: the living simulation. Today, many simulations require a wide range of numerical solvers and run most efficiently if specialized nodes are matched to the solvers. The idea of the living simulation is that it decides itself which grid machines to use based on the numerical solver currently in use.

1 Introduction

A grid application consists of a range of tasks, each of which may run most efficiently using a different set of resources. Most of these applications, however, use a fixed resource topology even though certain tasks could benefit from using different resources. This can be due to the computational demands of these tasks or due to a change in resource availability over time. A wide range of work has been done on developing external management systems that allow applications to change grid resources during execution. This includes workflow systems [1,2,3] or grid schedulers with migration capabilities [4,5] that support resource switches that are either part of a predefined workflow or requested by the user.

An application management system that autonomously switches at run-time has been proposed by [6], where a hierarchically distributed application management system dynamically schedules and migrates a bag-of-tasks style MPI application, using a static hierarchy of schedulers to accomplish this.

A self-adaptive grid application that does not require external managers has been presented in [7]. Although this application does not use grid scheduling,

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 205–212, 2009.

it is able to autonomously migrate to different locations and change its number of processes. This has been accomplished by allowing all processes to share knowledge and cooperate in managing the application's topology.

In this work, we propose the living grid application, in which the application also decides where to run, and which is also able to migrate itself at run-time to another computer when needed. The intelligent migration from one computer to another can be realized over a long baseline, but does not need to be designed this way (see Sec. 2). We have applied this method to a multi-scale simulation on an intercontinental grid of semi-dedicated computers.

2 Living Application

2.1 Rationale

A flexible approach is needed to execute a complex grid application with multiple tasks and a diverse palette of resource requirements. The application should then be able to switch between tasks at run-time and between the resources required for each of these tasks, while maintaining the integrity of its data during these switches.

A switch requires the application to terminate its current execution, output its current state, and from that reinitialize the application using a new resource topology suited for the task at hand. Previously this has been done on a grid only in orchestration with a workflow manager. A job submitted by a workflow manager lacks the ability to change its resource topology during execution, as it does not have the privileges to make use of grid schedulers. When running an application with multiple tasks, this results in a 'bouncing' pattern where the manager submits jobs which return once a switch is required, only to be instantly submitted again to handle a different task. In the most favorable case, the performance loss introduced by bouncing and managerial overhead can be limited, but even then the successful completion of the simulation depends on the availability of an external manager, which is a potential single point of failure.

2.2 How the Living Application Works

The living application switches between sites and tasks dynamically and without external dependencies. It is based on four principles:

1. It makes decisions on which tasks to do and which resources to use.
2. It makes these decisions based on knowledge it has acquired at run-time.
3. It changes resources and switches between tasks.
4. It operates autonomously.

As a living application operates autonomously on the grid, it obtains its privileges on its own without interacting with an external workflow manager or user.

Upon initialization, the application is locally equipped with the tools and data to perform the required tasks and the criteria for switching between tasks

or resource topologies. It is then submitted as a job to the grid with the initial resource requirements defined by the launcher. The living application begins execution on the grid and continues to do so until either a switch or a termination is required.

The conditions for switching or termination are determined prior to the start of the calculation or during run-time, but they are not necessarily static. They can rely on the internal state of the application, or on information from external sensors. When the conditions for a switch have been met, the application will migrate to different grid resources, switch to a different task, or both.

The switching between tasks requires two steps, which are finalizing the old task (and any program it still uses) and starting up the new task. During this switch, the application-specific data should be left intact. The switching between sites requires a larger number of actions, which are:

1. Creating a set of files consisting of the current application, files with its parameters and data and a script that specifies the methods and conditions for switching and termination.
2. Creating a job definition for the application on the new resources.
3. Authenticating (independently) on the grid.
4. Transferring the files to the remote site (if this is not done automatically by a resource broker).
5. Submitting the job, either through a resource broker or by directly accessing the head nodes of grid sites.
6. Reinitializing the living application on the new site.

Additional file transfer may be required, if the application has locally written data that is required elsewhere. The application could initiate the transfer of output files either during run-time (e.g. if separate files are written) or just before a job terminates on one machine (if data is appended to a single large file or data transfer would cause overhead at run-time).

The living application requires some user privileges to initiate data transfers and to autonomously migrate from one site to another. We obtain these privileges by using a grid client interface to access a credential management service. The details of this method are discussed in Sec. 2.3. The application requires access to the grid client interfaces on all participating nodes to request these privileges during execution. Once these privileges are granted, the application can perform authentication, data transfers and job submissions to the grid.

2.3 Security Considerations

User privileges on the grid are provided by an X.509 grid proxy [8] which requires the presence of a certificate, a private key and a correct pass phrase typed in by the user. This proxy is represented by a temporary file with limited lifetime. The easiest way to provide user privileges to a living application would be to equip it with this file, transporting it as it migrates, allowing it to reuse the proxy on remote locations. However, this approach has three drawbacks:

First, the presence of a proxy file on a remote site poses a security risk. If the file is not read-protected or stored in a shared account, it may be possible for other grid users to copy the proxy. The possession of this proxy enables them to impersonate the living application user for the duration of the proxy's lifetime, providing them with rights and resources that they could otherwise not use. Even if the proxy is on a dedicated account and read-protected, local users with admin rights are able to copy it and use it for impersonation.

Second, it is not possible to cancel the application after the first stage, as the proxy is initialized only at startup, after which it travels around on remote sites. This may cause a malfunctioning application to continue running and migrating until the proxy lifetime is exceeded. An application that is equipped for self-reproduction may iteratively spawns multiple successors which could lead to a grid meltdown.

Third, for the same reasons as before it is also not possible to prolong the lifetime of the proxy. This could cause the application to terminate prematurely once the proxy lifetime is exceeded. Specifying an excessively long lifetime relieves this problem, at the expense of increasing exposure to the other two drawbacks.

To reduce these drawbacks we have chosen to use an intermediary MyProxy server [9] in our implementation. The user initializes his or her proxy on the MyProxy server and defines a unique password. External sources use this password to access the server and initialize credentials with a limited life-time (normally set to 12 hours). The living application is provided with this password, and can therefore obtain these user privileges. If the password is stolen, others may be able to get the same privileges, but the user can block further access at any time by destroying the credential.

During application execution, the user can also extend the lifetime of his MyProxy credential by renewing it. It is also possible to replicate the credentials to other MyProxy servers, which allows the application to use remote MyProxy servers if the local server has died, rather than terminating itself upon switching.

2.4 Living Simulation

A special case of the living application is the living simulation. Today, simulations of complex systems, in which the dynamic range exceeds the standard precision of the computer, call for a wide range of numerical solvers [10]. Each of these solvers may run most efficiently on a different computer architecture. Most such simulations, however, are run on a single computer even though they would benefit from running on a variety of architectures.

This can be solved by migrating the application at run-time from one computer to another, in other words, by creating a living simulation. We demonstrate the concept of the living application by applying it to the (living) simulation of two galaxies merging.

The term living simulation has been previously defined as simulations that fine-tune their behavior at run-time based on input from external sensors, e.g. to provide input for performing adaptive load balancing [11]. In our definition we provide the simulation with user privileges and expect it to function autonomously.

3 Discussion

A living simulation is based on the principle that it autonomously switches between sites and solvers when required. This switching is done dynamically and without external dependencies. The simulation is locally equipped with the required solvers, the switching criteria and the initial conditions. It is then submitted as a job to the grid with the initial resource requirements defined by the launcher. The living simulation begins calculating on the grid and continues to do so until either a switching condition or a termination condition has been met.

By using the idea of the living applications, we have implemented and tested a living simulation, in which the merger of two galaxies, each with a central

Table 1. Specifications for the test nodes. The first column gives the name of the computer followed by its country of residence (NL for the Netherlands, US for the United States). The subsequent columns give the type of processor in the node, followed by the amount of RAM, the operating system, and the special hardware installed on the PC. Both nodes are connected to the internet with a 1 Gbit/s Ethernet card.

name	location	CPU type	RAM [MB]	OS	hardware
darkstar	NL	Core2Duo 3.0GHz	2048	Debian	Nvidia 8800 Ultra
zonker	US	2x Xeon 3.6GHz	2048	Gentoo	GRAPE 6A

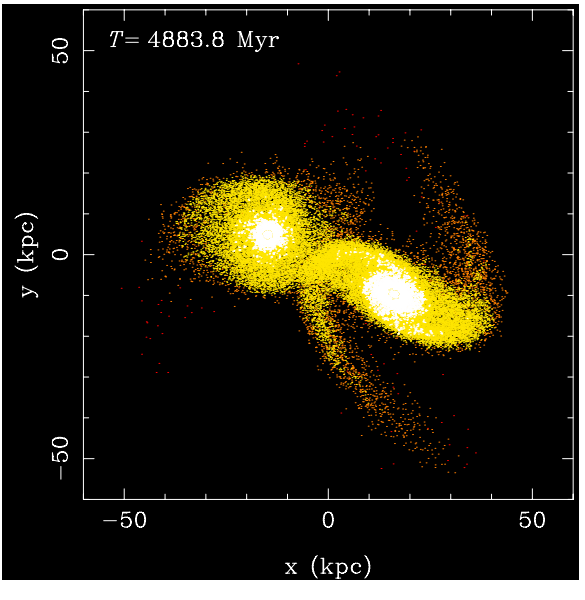

Fig. 1. Simulation snapshot of one of the runs, where the two galaxies approach for an initial interaction

supermassive black hole (SMBH), is simulated. We used a GPU-enabled tree code [12,13] for the early stages of the merger and switched to a GRAPE-enabled direct integrator [14] during later stages, when the separation between the two SMBHs was sufficiently small. We tested the overhead of our simulation using two machines with Globus middleware, one of which was equipped with GRAPE dedicated hardware (GRAvity PipE, [15]) and the other with a Nvidia 8800 GTX Ultra GPU. A specification for both nodes can be found in Tab. 1.

During our living simulation runs, the simulation performed three switches. The overhead caused by autonomously switching between machines was marginal, amounting to $\sim$ 4 percent of the three hours it for a 64k particle run to complete. A sample snapshot from on of our runs can be found in Fig. 1.

4 Conclusion

We introduced the living application as a way to manage complex applications on a large distributed infrastructure. Due to the autonomous nature of a living simulation, it is important to provide a mechanism that allows the user to terminate it. By having the simulation retrieve its extended privileges from a credential management service (MyProxy), users are able to revoke the privileges of the simulation regardless of its location. In addition, we can make repeated use of short-lived proxy credentials instead of a a single long-lived credential, which poses a larger security risk.

We have applied this concept in a living simulation of two galaxies merging. Our approach allows the simulation to use the optimal compute resources for each of the two solvers, switching resources whenever a different solver is required. In our example case, the solvers were a tree code and a direct N-body method, which were optimized for two kinds of special-purpose hardware, namely a GPU (tree) and a GRAPE (direct). The switches between these two solvers take place without user intervention, remote output retrieval or external managers. The execution time was only affected marginally by overhead such as caused by job migration and data transfer over the grid.

The creation of grid species enables us to give a simulation the ability to autonomously use the grid, acquire and apply internal knowledge, and migrate themselves. If we expand the awareness of a living simulation by letting it inherit more advanced sensing and scheduling abilities, we will be able to apply it to problems of greater complexity. In that way we could allow our simulation to evolve to a more complex organism.

Acknowledgements

We are grateful to Rick Quax for his contributions during the early stages of this project, and Steve McMillan for technical support and access to the GRAPE at Drexel University. We also thank David Groep and Hashim Mohammed for

their input on the security aspects of this work, to Evghenii Gaburov for sharing his octgrav code for our experiments and to Alfons Hoekstra for his useful suggestions.

This research is supported by the Netherlands organization for Scientific research (NWO) grant #643.200.503, the European Commission grant for the QosCosGrid project (grant number: FP6-2005-IST-5 033883), the Netherlands Advanced School for Astronomy (NOVA) and the Leids Kerkhoven-Bosscha fonds (LKBF).

References

1. Herrera, J., Huedo, E., Montero, R., Llorente, I.: Porting of scientific applications to grid computing on gridway. Sci. Program. 13(4), 317–331 (2005)
2. Ludscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency and Computation: Practice and Experience 18(10), 1039–1065 (2006)
3. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. Journal of Grid Computing 3, 171–200 (2005)
4. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-g: A computation management agent for multi-institutional grids. Cluster Computing 5, 237–246 (2002)
5. Allen, G., Angulo, D., Foster, I., Lanfermann, G., Liu, C., Radke, T., Seidel, E., Shalf, J.: The cactus worm: Experiments with dynamic resource discovery and allocation in a grid environment. Int. J. High Perform. Comput. Appl. 15(4), 345–358 (2001)
6. Nascimento, A., Sena, A., Boeres, C., Rebello, V.: Distributed and dynamic self-scheduling of parallel mpi grid applications. Concurr. Comput.: Pract. Exper. 19(14), 1955–1974 (2007)
7. Wrzesinska, G., van Nieuwpoort, R., Maassen, J., Bal, H.: Fault-tolerance, malleability and migration for divide-and-conquer applications on the grid. In: IPDPS 2005: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005) - Papers, Washington, DC, USA. IEEE Computer Society, Los Alamitos (2005)
8. Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Gawor, J., Meder, S., Siebenlist, F.: X.509 proxy certificates for dynamic delegation. In: Proceedings of the 3rd Annual PKI R&D Workshop (2004)
9. Basney, J., Humphrey, M., Welch, V.: The myproxy online credential repository. Software: Practise and Experience 35(9), 801–816 (2005)
10. Hoekstra, A., Portegies Zwart, G., Bubak, S., Sloot, P.: Towards Distributed Petascale Computing. In: Bader, D.A. (ed.) Petascale Computing: Algorithms and Applications. Computational Science Series, 565 pp. Chapman & Hall/CRC, Boca Raton (2008)
11. Korkhov, V.V., Krzhizhanovskaya, V.V., Sloot, P.M.A.: A grid-based virtual reactor: Parallel performance and adaptive load balancing. J. Parallel Distrib. Comput. 68(5), 596–608 (2008)
12. Barnes, J., Hut, P.: A Hierarchical O(NlogN) Force-Calculation Algorithm. Nature 324, 446–449 (1986)

13. Gaburov, E., Nitadori, K., Harfst, S., Portegies Zwart, S., Makino, J.: A gravitational tree code on graphics processing units: Implementation in cuda (in preparation) (2009)
14. Harfst, S., Gualandris, A., Merritt, D., Spurzem, R., Portegies Zwart, S., Berczik, P.: Performance analysis of direct N-body algorithms on special-purpose supercomputers. New Astronomy 12, 357–377 (2007)
15. Sugimoto, D., Chikada, Y., Makino, J., Ito, T., Ebisuzaki, T., Umemura, M.: A Special-Purpose Computer for Gravitational Many-Body Problems. Nature 345, 33–35 (1990)