2015-10

FriendComputing: Organic application centric distributed computing

Wolf, Beat

Carretero Pérez, Jesús; et.al. (eds.). (2015) Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015): Krakow, Poland. Universidad Carlos III de Madrid, pp. 117-119.
http://hdl.handle.net/10016/22003

Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)
Krakow, Poland

Jesus Carretero, Javier Garcia Blas
Roman Wyrzykowski, Emmanuel Jeannot.
(Editors)

September 10-11, 2015

FriendComputing: Organic application centric distributed computing

Beat Wolf, Loïc Monney, Pierre Kuonen

University of Applied Sciences Western Switzerland, HES-SO//Fribourg
beat.wolf@hefr.ch

Abstract

Building Ultrascale computer systems is a hard problem, not yet solved and fully explored. Combining the computing resources of multiple organizations, often in different administrative domains with heterogeneous hardware and diverse demands on the system, requires new tools and frameworks to be put in place. During previous work we developed POP-Java, a Java programming language extension that allows to easily develop distributed applications in a heterogeneous environment. We now present an extension to the POP-Java language, that allows to create application centered networks in which any member can benefit from the computing power and storage capacity of its members. An accounting system is integrated, allowing the different members of the network to bill the usage of their resources to the other members, if so desired. The system is expanded through a similar process as seen in social networks, making it possible to use the resources of friend and friends of friends. Parts of the proposed system has been implemented as a prototype inside the POP-Java programming language.

Keywords Java, distributed computing

I. Introduction

Scientific and commercial applications face an increase of computational resource requirements. This can be observed in various domains, such as medical research applications, material simulations, weather forecasts or multimedia processing. In all those domains the data to be analysed is produced at increasing speeds, with data processing applications not being able to keep up with the analysis. In the past, mainly large organizations have addressed these types of analysis as they had access to large computing infrastructures. The constant improvements of computers as well as the reduction of their prices, has attracted the interest of small organizations to tackle this type of calculations. Nevertheless, the constantly increasing amounts of data produced as well as the complexity of the types of analyses to perform still restricts many small organizations to really address this domain.

Several technologies emerged over the past years to assist these organizations to cope with the demands of modern applications. Especially, technologies like cloud computing enabled these organizations to expand their computing infrastructure for cases where it is not sufficient. While for many organizations this is an acceptable solution, it is not a solution for all use cases. This is especially true for organizations working with sensitive data, for example medical data, that may not be allowed to be sent to any remote location. This problem is even more emphasized when the cloud provider is located in a different jurisdiction, a likely scenario in regards to currently popular cloud providers like Amazon.

Various commercial and academic computing grids have been created in the last decades. They regroup the infrastructures of several organizations into a single grid, available for use to all members. Grids such as the Open Science Grid (OSG) [1] or the Worldwide LHC Computing Grid (WLCG) [2] come in the form of traditional grids which can be used for a multitude of applications by various users. Other grids, like Folding@Home [3] and the BOINC [4] based grids are

application specific, generally used by a single user providing the analysis tasks.

The main drawback of the approach of the grid is due to the fact that the code must be adapted to different operating systems and hardware present in the grids. This leads to costly IT developments that are often beyond the capacity of these small organizations. This problem has been largely reduced in the clouds thanks to the virtualization.

In addition, the setup and maintenance of a grid environment is a complicated task and many organizations do not have the necessary technical knowledge to do so. A use case which is quite common is that multiple partners need a significant amount of computing power to perform the same type of analysis. In such a situation the partners often use the same software to perform their analysis and could benefit from using each other's infrastructure to do so. It is this particular use case that we address in this poster.

II. Friend computing

The concept of friend computing is to create an application centered network of so called "friends", which share the same goal. Multiple users of a certain software which performs computationally complex analyses can group together and benefit from each other's infrastructure. The group is expanded through a process similar to how social networks work. Any new member gets invited by an existing member and once part of the network can access the computing power of every other installation. This approach is similar to the friend-to-friend computing as presented in [5], with the main difference that in [5], authors focus on data sharing only. In [6] the authors have shown that friend-to-friend computing can also been applied for sharing computing resources, an idea on which we expand upon.

We based our first prototype implementation of friend computing on POP-Java [9], an extension of the Java programming language which implements the POP (Parallel Object Programming) model [7]. The POP programming model was initially implemented as an extension of the C++ language, called POP-C++ [8]. POP-Java was chosen as it offers an excellent base for the concept of friend computing.

The POP-Java language has as one of its main features the possibility to create objects on remote computers, making it possible for the programmers to combine local and remote objects. By default POP-Java will either use any available computer in a locally configured POP-Java network, or a specific computer defined by the programmer. The introduction of Friend computing allows the programmer to search for a computing resource in the friend network automatically, with the ability to specify certain criteria such as processing power or storage space. This can also be used to bring calculations to the place where the data is stored, which can reduce privacy concerns in cases where the data itself is sensitive.

The concept of friend computing was also approached from a commercial viewpoint, giving the users of a network an incentive to make their resources available. Because of this an accounting system was integrated, in which every member of the network logs the usage of their resources by other members. This makes it possible to bill the different users of the network based on their usage, increasing the incentive to participate in the network as well as giving the possibility to monetize their infrastructure.

III. Prototype

We created a prototype of the concept of friend computing using POP-Java. It consists of an extended version of the POP-Java language, as well as an example application using those features. This prototype application was used to verify the correct implementation of the POP-Java implementation which includes the friend computing extension.

The prototype allows the creation of new friend networks, allowing to define an ID and purpose of the friend computing network. This newly created network can then be extended by inviting new members to the network. To get invited, the joining party needs to provide an IP address to a member of the network and have the prototype application running. The prototype application will show a notification of the network invitation, and upon accepting sets up the required friend computing network configurations. Once a member of the network, any member can launch a simple calculation, in this case the factorization of a

number. When the calculation is launched, available resources inside the friend computing network are automatically discovered and the calculations distributed over multiple computers.

Any usage of the resources is journaled and can later be used to bill the individual members of the network (if so wished).

IV. Conclusion

We showed a concept on how to approach distributed software development in an Ultrascale world. The ability to dynamically grow a distributed computing network based around a specific application with the possibility to bill other members based on their usage is an interesting approach to use ultrascale systems. Our current prototype has only been verified on a small scale network and further tests will show how it scales to larger numbers. Further work will be required to scale the current concept to Ultrascale systems, but the current concept already allows for the design of very large distributed applications. This is especially true if like in every object orientated application, the scope of every object is limited as much as possible. This reduces not only the complexity of the application, but also the complexity of the network traffic between the different distributed objects.

Other open works include the improvement of the resource discovery protocol. While the currently used resource discovery protocol, which is the one used by POP-Java, works well when searching for resources with a certain amount of RAM or CPU power, in the context of friend computing it would be helpful to be able to search for computing nodes which have certain data stored locally. This would allow bringing the computations to the data instead of the other way around, greatly reducing data confidentially issues that can arise in distributed systems.

The prototype and POP-Java in general also does not yet handle firewalls and systems behind a NAT. Including support for those would greatly help adoption on a larger scale including commercial applications like GensearchNGS [10], an genetic diagnostics application which served as the initial inspiration for this project.

References

[1] R. Pordes, *et al.. The open science grid.* Journal of Physics: Conference Series. Vol. 78. No. 1. IOP Publishing, 2007

[2] D. Bonacorsi and T. Ferrari, *WLCG Service Challenges and Tiered architecture in the LHC era.*, IFAE 2006. Springer Milan, 2007. 365-368.

[3] S. M.Larson, *et al.*, *Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology.* 2002

[4] D.P. Anderson, *Boinc: A system for public-resource computing and storage*, Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing. IEEE, 2004

[5] B. Popescu, *et al.*, *Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System*, Security Protocols, 2006

[6] U.Norbisrath, *et al.*, *Friend-to-friend computing instant messaging based spontaneous desktop grid*, Proceedings - 3rd International Conference on Internet and Web Applications and Services, ICIW 2008

[7] T. A.Nguyen and P. Kuonen, *A model of dynamic parallel objects for metacomputing*, The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications, 2002.

[8] C. D.Jiogo *et al.*, *Parallel Object Programming in POP-C++: A Case Study for Sparse Matrix-vector Multiplication*, Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP06), France, 2006

[9] B. Wolf *et al.*, *POP-Java : Parallélisme et distribution orienté objet*, ComPAS 2014 : confÃl'rence en parallélisme, architecture et systèmes, 2014

[10] B. Wolf *et al.*, *DNAseq Workflow in a Diagnostic Context and an Example of a User Friendly Implementation*, BioMed Research International, vol. 2015