Implementing Abstractions for Data Intensive Applications using SAGA

Shantenu Jha[1,2], Hartmut Kaiser[1], Michael Miceli[1], Christopher Miceli[1], Joao Abecasis[1]

[1]*Center for Computation & Technology, Louisiana State University, USA*

[2]*Department of Computer Science, Louisiana State University, USA*

May 15, 2008

The deluge of data is upon us and there exist a critical need to be able to access, manage and mine this data. There is a need for abstractions to support data-intensive computing, and these abstractions are required at several levels – programmatic, system and data-access patterns.

For example, the power of Google is based upon a simple programming abstraction – MapReduce [1]. MapReduce is a framework for splitting up large data and keeping track of which machine the data is on. The purpose is to automatically parallelize and execute on large clusters. The application itself has to provide only the specific map and reduce steps, reusing the bulk of the functionality from the provided generic implementation. By relieving the end-developer from having to control explicitly data placement, job distribution, load balancing, etc., MapReduce provides a level of abstraction that simplifies the task of an application programmer.

There exist additional powerful abstractions, such as AllPairs [2], which is both a programming abstractions and a system-level abstraction. Allpairs is a framework for when every element of some data needs every other element of another set of data. When implemented on distributed infrastructure, AllPairs, like MapReduce, essentially decouples the client's code from grid specific information. Again, all the application has to provide is the ordering criteria of the data to work on, resuing the programming abstractions as implemented by the based framework.

SAGA [3] is a high level API that provides a simple, standard and uniform interface to the most commonly required distributed functionality. SAGA can be used to encode grid applications [4, 5], tool-kits to manage distributed applications as well as implement abstractions that support commonly occurring programming, access and usage patterns. The focus of this paper is on the latter set, i.e. the use of SAGA in implementing well known abstractions for data intensive computing.

In this paper, we will implement MapReduce and All-Pairs abstractions using SAGA and use them to solve commonly encountered genomic tasks. We will show how multiple sequence alignment can be orchestrated using the SAGA-Allpair implementation, and genome searching can be implemented using Map-reduce. In addition, the aim of this paper is to show (validate) that SAGA is a sufficiently complete and high-level interface so as to support these programming abstractions.

Figure 1 illustrates the software architecture of the implementation, highlighting the different abstraction levels that allow the reuse of most of the system for both algorithms and for different genomic applications. We will highlight the sailent points of our implementations, and how we handle common considerations such as when to move the data to the machine or when to process it locally. The implemention of these abstractions encapsulates details such as latency hiding, performance and other variables (such as cluster sizes, and queue sizes). The user should be able to easily add a few function calls without worrying about many considerations required by most grid computing applications.

We will discuss other performance issues that arise when implementing abstractions specific for data-intensive computing. A grid application's design should not focus on the bandwidth of the network, the dispatch latency, the number of machines available, and data reliability. Even something as simple as process size can be a tough challenge to optimize. If a job is too small, then network traffic becomes a bottleneck and the design is inefficient. If a job is too large, it is difficult to tell when it is hanging or still computing. Also, if another job with a higher priority takes a machine over, the application will be waiting on jobs longer. The main point of this paper is to show how a flexible, extensible implementation of programming data-intensive abstractions using SAGA can shield the application developer many of these considerations.

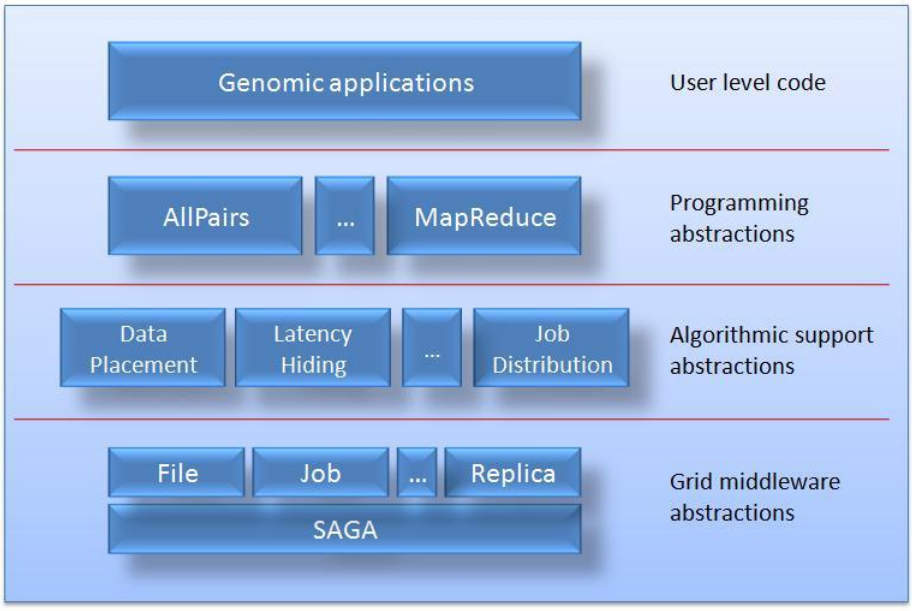

Figure 1: The programming abstractions provided by the AllPairs and MapReduce algorithm implementations are sufficient to enable a wide range of genomic aplications (sequence alignment, searching, etc.) to be distributed over grids. The required underlying functionality is provided by higher level algorithmic support abstractions, such as tools for data placement, latency hiding, and job distribution, which ar ebeing built on top of grid middleware abstractions provided by SAGA (such as file transfer, job launching, information services and replica management.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 137–150.

[2] All-Pairs: An Abstraction for Data Intensive Cloud Computing, Christopher Moretti, Jared Bulosan, Douglas Thain, and Patrick Flynn, IEEE International Parallel and Distributed Processing Symposium (IPDPS), April 2008.

[3] OGF Document Series 90, http://www.ogf.org/documents/GFD.90.pdf.

[4] S. Jha, H. Kaiser, Y. El Khamra, and O. Weidner, "Design and implementation of network performance aware applications using SAGA and Cactus," in *Accepted for 3rd IEEE Conference on eScience2007 and Grid Computing, Bangalore, India.*, 2007. [Online]. Available: http://saga.cct.lsu.edu/publications/saga_cactus_escience.pdf

[5] Developing Large-Scale Adaptive Scientific Applications with Hard to Predict Runtime Resource Requirements, *Proceedings of TeraGrid08*, available at http://tinyurl.com/5du32j.