Understanding Application-Level Interoperability: Scaling-Out MapReduce Over High-Performance Grids and Clouds

Saurabh Sehgal[1], Miklos Erdelyi[3,4], Andre Merzky[1], Shantenu Jha[*1,2]
[1] *Center for Computation & Technology, Louisiana State University, USA*
[2] *Department of Computer Science, Louisiana State University, USA*
[3] *Department of Computer Science & Systems Technology, University of Pannonia, Veszprem, Hungary*
[4] *Computer & Automation Research Institute of the Hungarian Academy of Sciences*
[*] *Contact Author* sjha@cct.lsu.edu

Abstract

Application-level interoperability is defined as the ability of an application to utilize multiple distributed heterogeneous resources. Such interoperability is becoming increasingly important with increasing volumes of data, multiple sources of data as well as resource types. The primary aim of this paper is to understand different ways and levels in which application-level interoperability can be provided across distributed infrastructure. Our approach is: (i) Given the simplicity of MapReduce, its wide-spread usage, and its ability to capture the primary challenges of developing distributed applications, use MapReduce as the underlying exemplar; we develop an interoperable implementation of MapReduce using SAGA – an API to support distributed programming, (ii) Using the canonical wordcount application that uses SAGA-based MapReduce, we investigate its scale-out across clusters, clouds and HPC resources, (iii) Establish the execution of wordcount application using MapReduce and other programming models such as Sphere concurrently. SAGA-based MapReduce in addition to being interoperable across different distributed infrastructures, also provides user-level control of the relative placement of compute and data. We provide performance measures and analysis of SAGA-MapReduce when using multiple, different, heterogeneous infrastructures concurrently for the same problem instance.

1. Introduction

There are numerous scientific applications that either currently utilize, or need to utilize data and resources distributed over vast heterogeneous infrastructures and networks with varying speeds and characteristics. Many scientific applications are, however, designed with a specific infrastructure; such dependence and tight-coupling to specific resource types and technologies is, in a heterogeneous distributed environment, not an optimal design choice. In order to leverage the flexibility of distributed systems and to gain maximum run-time performance, applications must shed their dependence on single infrastructure for all of their computational and data processing needs. For example, the Sector/Sphere data cloud is exclusively designed to support data-intensive computing on high speed networks, while others, like the distributed file systems GFS/HDFS, assume limited bandwidth among infrastructure nodes [14, 10]. Thus, for applications to efficiently utilize heterogeneous environments, abstractions must be developed for the efficient utilization of and orchestration across such distinct distributed infrastructure.

In addition to issues of performance and scale addressed in the previous paragraph, the transition of existing distributed programming models and applications to emerging and novel distributed infrastructure must be as seamless and as non-disruptive as possible. A fundamental question at the heart of all these considerations is the question of how scientific applications can be developed so as to utilize as broad a range of distributed systems as possible, without vendor lock-in, yet with the flexibility and performance that scientific applications demand.

We define Application Level Interoperability (ALI) as a feature that arises, when other than say compiling, there are no further changes required of the application to utilize a new platform. The complexity of providing ALI varies and depends upon the application under consideration. For example, it is somewhat easier for simple "distribution unaware" applications to utilize multiple het-

erogeneous distributed environments, than for applications where multiple distinct and possibly distributed components need to coordinate and communicate.

The Case for Application-level Interoperability. In either case, ALI is not only of theoretical interest. There exist many applications which involve large volumes of data on distributed heterogeneous resources and which benefit from ALI. For examples, the Earth System Grid [1] involves peta to exa-bytes of data, and one thus cannot move all data (given current transfer capabilities), nor compute at a centralized location. Thus there is an imperative to operate on the data *in situ*, which in turn involves computation across heterogeneous distributed platforms as part of the same application.

In addition, there exist a wide range of applications that have decomposable but heterogeneous computational tasks. It is conceivable, that some of these tasks are better suited for traditional grids, whilst some are better placed in cloud environments. The LEAD application, as part of the VGrADS project provides a prominent example[1]. Due to different data-compute affinity requirement amongst the tasks, some workloads might be better placed on a cloud [2], whilst some may optimally be located on regular grids. Complex dependencies and inter-relationships between sub-tasks make this often difficult to determine before run-time.

Last, but not least, in the rapidly evolving world of clouds, there is as of yet little business motivation for cloud providers to define, implement and support new/standard interfaces. Consequently, there is a case to be made that by providing ALI, such barriers can be overcome and cross-cloud applications can be easily achieved.

Currently, many programming models and abstractions are tied to a specific back-end infrastructure. For example, Google's MapReduce [12], which is tied to Google's file system, or Sphere[15] which is linked to the Sector file system. It is often the case that an application maybe significantly better suited to a specific programming model; similarly a specific programming model maybe optimised for a specific infrastructure. However, where this is not necessarily the case, or more importantly, when different applications or programming models can utilise "non-native" infrastructure, the ability to mix-and-match across the layers of applications, programming models and infrastructure should be supported. Ideally, an application should be able to utilize any programming model, and any programming model should be executable on any underlying infrastructure. Thus there is a need to investigate interoperability of different programming models for the same application on different systems.

We will work with MapReduce and an application based on MapReduce– the canonical wordcount application. We use SAGA — Simple API for Grid Applications" (see Sec. 2) as the programming system for distributed applications. In Ref. [18], we implemented a MapReduce based wordcount application using SAGA. We demonstrated that the SAGA-based implementation is infrastructure independent, whilst still providing control over deployment, distribution, run-time decomposition and data/compute co-location. We demonstrated that SAGA-MapReduce is interoperable on traditional (grids) and emerging (clouds) distributed infrastructure *concurrently and cooperatively towards a solution of the same problem instance.*

The primary focus of this paper is to understand, demonstrate and investigate different types of ALI. We build upon and use SAGA-based MapReduce as an exemplar to discuss multiple levels and types of interoperability that can exist between infrastructures. We use SAGA-MapReduce and SAGA-based Sphere on different infrastructure. We will also show that our approach to ALI helps break the coupling between programming models and infrastructure on the one hand, whilst providing empirically-driven insight about the performance of an application with different programming models.

This paper is structured as follows: Section 2 gives a short overview over those SAGA extensions which enable specifically the ALI work discussed in this paper. Section 3 describes our SAGA-MapReduce implementation. Section 4 discusses the different levels of ALI we investigate and demonstrate, with more details on the experiments provided in Section 5. Section 6 concludes the paper with a discussion of the results.

2. SAGA

The SAGA [19, 16] programming system provides a high level API that forms a simple, standard and uniform interface for the most commonly required

[1] `http://vgrads.rice.edu/presentations/VGrADS_overview_SC08pdf.pdf`