P*: A Model of Pilot-Abstractions

Andre Luckow, Ole Weidner,
Andre Merzky, Sharath Maddineni
Center for Computation and Technology
Louisiana State University
Baton Rouge, LA
{aluckow, oweidern, amerzky,
smaddini}@cct.lsu.edu

Mark Santcroos
Bioinformatics Laboratory
Academic Medical Center
University of Amsterdam
Amsterdam, The Netherlands
m.a.santcroos@amc.uva.nl

Shantenu Jha*
CAC/ECE
Rutgers University
Piscataway, NJ
shantenu.jha@rutgers.edu

ABSTRACT

Pilot-Jobs (PJ) have become one of the most successful abstractions in distributed computing. In spite of extensive uptake, there does not exist a well defined, unifying conceptual model of Pilot-Jobs, which can be used to define, compare and contrast PJ implementations. This presents a barrier to extensibility and interoperability. This paper is an attempt to, (i) provide a minimal but complete model (P*) of Pilot-Jobs, (ii) establish the generality of the P* Model by mapping various well-known Pilot-Job frameworks such as Condor and DIANE to P*, (iii) demonstrate the interoperable and concurrent usage of *distinct* pilot-job frameworks on different production distributed cyberinfrastructures via the use of an extensible API for the P* Model (Pilot-API).

Categories and Subject Descriptors

D.2.10 [**Software Design**]: Methodologies

General Terms

Distributed Systems, Interoperability, Abstractions

1. INTRODUCTION AND OVERVIEW

Distributed cyber/e-infrastructure is by definition comprised of a set of resources that is fluctuating – growing, shrinking, changing in load and capability (in contrast to a static resource utilization model of traditional cluster computing systems). The ability to utilize a dynamic resource pool is thus an important attribute of any application that needs to utilize distributed cyberinfrastructure (DCI) efficiently. Pilot-Jobs (PJ) provide an effective abstraction for dynamic execution and resource utilization in a distributed context. As a consequence of providing a simple approach for decoupling workload management and resource assignment/scheduling, Pilot-Jobs have been one of the most successful abstractions in distributed computing. Not surprisingly, there are multiple, distinct and incompatible implementations of Pilot-Jobs. Often, these implementations are coupled to the infrastructure they were designed for.

Our objective is to provide a minimal, but complete model of Pilot-Jobs: The P* Model provides a conceptual basis to compare and contrast different PJ frameworks. We perform experiments demonstrating interoperability across middleware, platform and different PJ frameworks, and to establish the effectiveness of the Pilot-API.

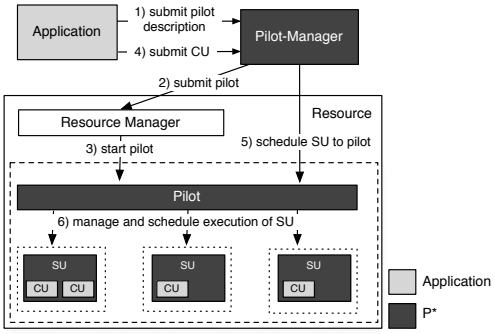

Figure 1: P* Model: Elements and Interactions: The manager has two functions: it manages 1) Pilots (step 1-3) and 2) the execution of CUs. After a CU is submitted to the manager, it transitions to an SU, which is scheduled to a Pilot by the PM.

2. THE P* MODEL OF PILOT-ABSTRACTIONS

The P* model is derived from an analysis of many PJ implementations; based upon this analysis, we present the common *elements* of the P* Model, followed by a description of the overall functioning of a PJ framework. The P* Model defines the following elements:

• **Pilot (Pilot-Compute):** The Pilot is the entity that actually gets submitted and scheduled on a resource. The PJ provides application (user) level control and management of the set of allocated resources.

*contact author

HPDC'12, June 18–22, 2012, Delft, The Netherlands.

P* Element	BigJob	DIANE	Condor-G/Glide-in
Pilot-Manager	BigJob Manager	RunMaster	condor_master condor_collector condor_negotiator condor_schedd
Pilot	BigJob Agent	Worker Agent	condor_master condor_startd
CU	Task	Task	Job
SU	Sub-Job	Task	Job

Table 1: Mapping P* elements and PJ Frameworks: While each PJ framework maintains its own vocabulary, each of the P* elements can be mapped to one (or more) components of the different frameworks.

• **Compute Unit (CU):** A CU encapsulates a self-contained piece of work (a task) specified by the application that is submitted to the Pilot-Job framework. There is no intrinsic notion of resource associated with a CU.
• **Scheduling Unit (SU):** SUs are the units of scheduling, internal to the P* Model, i.e., it is not known by or visible to an application. Once a CU is under the control of the Pilot-Job framework, it is assigned to an SU.
• **Pilot-Manager (PM):** The PM is responsible for (i) orchestrating the interaction between the Pilots as well as the different components of the P* Model (CUs, SUs) and (ii) decisions related to internal resource assignment (once resources have been acquired by the Pilot-Job). For example, an SU can consists of one or more CUs. The PM determines how to group them and when SUs are scheduled and executed on a resource via the Pilot.

Figure 1 illustrates the interactions between the elements of the P* Model. First, the application specifies the capabilities of the resources required using a Pilot-Job description (step 1). The PM then submits the necessary number of Pilots to fulfill the resource requirements of the application (step 2). Each Pilot is queued at the resource manager, which is responsible for starting it (step 3). The application can submit CUs to the PM at any time (step 4). A submitted CU becomes an SU, i. e. the PM is now in control of it. In the simplest case one CU corresponds to one SU.

As shown in table 1, the above elements can be mapped to specific entities in different Pilot-Jobs framework, e. g. BigJob [1], Condor [2] and DIANE [3]. While most of these frameworks share many properties, they often differ in their implementation (e. g. of the communication mechanism) and usage modalities. The aim of the Pilot-API [4] is to provide an abstract, unified interface to PJ frameworks that adhere to the P* Model. Figure 2 shows how the Pilot-API enables the user to run applications interoperably on different production and research infrastructures. For this purpose we investigate the performance of BFAST, a genome sequencing application. BFAST is very I/O sensitive – we observed for example, an I/O bottleneck if many BFAST CUs are run on the same shared file system. The Pilot-API enables applications to scale to different infrastructures in such cases.

3. DISCUSSION AND FUTURE WORK

Although a variety of PJ frameworks have emerged, which are, for the most parts, functionally equivalent, it is often impossible to use them interoperably or even just to compare them. The primary contribution of this work is the development of the P* Model, the usage of the P* elements to describe and characterize PJ frameworks such as DIANE and Condor-G/Glide-in. We demonstrated the practical relevance of this work by using different PJ frameworks via a unified API, and enabling applications to scale across different infrastructures and utilize performance advantages.

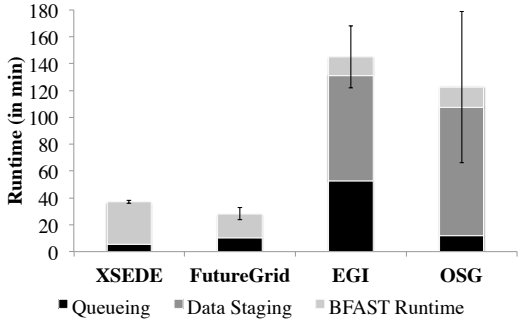

Figure 2: PJ Framework Performance on XSEDE, FutureGrid, EGI and OSG: Running 128 BFAST match tasks on 128 cores. The longer runtimes on EGI and OSG are mainly caused by longer queuing times and the necessity to stage all input files.

Pilot-abstractions provide significant future research & development opportunities, especially in their use for data-intensive applications. The management, placement and scheduling of data in distributed systems remains a challenge due to various reasons: (i) the placement of data is often decoupled from the placement of Compute Units and Pilots, i. e. the application must often manually stage in and out its data using simple scripts; (ii) heterogeneity, e. g. with respect to storage, filesystem types and paths, often prohibits late binding decisions; (iii) absence of capabilities that allow applications to specify their data dependencies on an abstract, logical level (rather than on file basis) are not available; (iv) due to lack of a common treatment for compute and data, optimizations of data/compute placements are often not possible. In addition, applications must cope with various other challenging, data-related issues, e. g. varying data sources (such as sensors and/or other application components), fluctuating data rates, transfer failures, optimizations for different queries, data-/compute co-location etc. While these issues can be handled in an application-specific manner, the use of general-purpose, infrastructure independent capabilities, such as a common Pilot-based abstraction for compute and data presents several advantages. This motivates an abstraction for data, that is analogous to Pilot-Jobs, that we refer to as *Pilot-Data (PD)*.

4. REFERENCES

[1] A. Luckow, L. Lacinski, and S. Jha, "SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems," in *IEEE/ACM CCGrid*, 2010, pp. 135–144.

[2] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002.

[3] J. Moscicki, "Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data," in *Nuclear Science Symposium Conference Record, 2003 IEEE*, vol. 3, 2003, pp. 1617 – 1620.

[4] Pilot API, https://github.com/saga-project/BigJob/blob/master/pilot/api/, 2012.