World Academy of Science, Engineering and Technology
International Journal of Industrial and Manufacturing Engineering
Vol:4, No:4, 2010

Generic Workload Management System Using Condor-Based Pilot Factory in PanDA Framework

Po-Hsiang Chiu, Torre Wenaus

***Abstract*—**In the current Grid environment, efficient workload management presents a significant challenge, for which there are exorbitant de facto standards encompassing resource discovery, brokerage, and data transfer, among others. In addition, the real-time resource status, essential for an optimal resource allocation strategy, is often not readily accessible. To address these issues and provide a cleaner abstraction of the Grid with the potential of generalizing into arbitrary resource-sharing environment, this paper proposes a new Condor-based pilot mechanism applied in the PanDA architecture, PanDA-PF WMS, with the goal of providing a more generic yet efficient resource allocating strategy. In this architecture, the PanDA server primarily acts as a repository of user jobs, responding to pilot requests from distributed, remote resources. Scheduling decisions are subsequently made according to the real-time resource information reported by pilots. Pilot Factory is a Condor-inspired solution for a scalable pilot dissemination and effectively functions as a resource provisioning mechanism through which the user-job server, PanDA, reaches out to the candidate resources only on demand.

***Keywords*—**Condor, glidein, PanDA, Pilot, Pilot Factory.

I. Introduction

In scientific community, many research and engineering projects over the past few years have gradually evolved to large-scale collaborations from different organizations through the use of geographically dispersed compute resources over the network. Projects that require such collaborative endeavor often involve cross-disciplinary settings with massive amount of data exchanged for the purpose of simulation and analysis such as Monte Carlo computing, large-scale optimization, and pattern discovery. Examples can be observed in ATLAS experiment [1-2], Human Genome Project [3], and SERENDIP [4], etc. To harness the distributed computational power, often the problem to solve is decomposed into several subtasks, whether it be independent user jobs or workflows containing several interdependent tasks (e.g. DAGMan [7]), followed by allocations to the desired compute resources for their results and feedback. Correspondingly, a user-friendly interface for task submissions, output retrievals and fast turn-around time are among the core issues to be considered.

Po-Hsiang Chiu is with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: po-hsiang.chiu@mavs.uta.edu).

Torre Wenaus is with the PAS Group at the Department of Physics, Brookhaven National Laboratory, Upton 11973 USA (e-mail: wenaus@bnl.gov)

Further, it is foreseeable that the conjoint effort of shared and distributed resources will potentially extend beyond the boundaries of current dedicated clusters and computing farms, as seen in the conventional Grid, to the inclusion of sensor network, personal computers, mobile computers, etc as evidenced by the growth of Volunteer Computing [9] and Opportunistic Computing [15]. Heterogeneity of distributed compute resources will thus become increasingly prominent. Nevertheless, even with the current Grid environment of modest scale with mostly clouds of computing farms, there already exist highly diversified infrastructures in the aspects of resource brokerage, data transfer, site services, and resource access and sharing mechanisms, among many others.

In light of the inherently complex and diversified resource-sharing environment, perhaps a more generic framework is necessary to accommodate potentially multiple forms of computing resources. The proposed new workload management system (WMS) is therefore conceived based upon the existing PanDA system [12] and further extended by a new pilot mechanism, i.e. *pilot factory*. For the intent of later discussion, the new WMS referenced above is conveniently termed PanDA-PF WMS. The PanDA architecture is built mostly on top of the existing networking infrastructure and database technology with a user-friendly and uniform interface to task submission and scheduling. The PanDA system works naturally with general pilot mechanism [14], a method of resource allocation where pilots are distributed to candidate resources to capture their real-time information, prepare and validate the computing environment before requesting real payloads of pending user jobs. In this manner, users need not be concerned about the details and differences in the underlying Grid infrastructure such as its associated scheduler and brokerage system; meanwhile, the real-time resource status collected by pilots allows for a more robust job scheduling and processing without unexpected failures resulted from, for example, inaccurate estimate of resource capacity (e.g. load average, CPU time, remaining memory, etc), incomplete software stack, or missing input datasets.

A. *Cross-Domain Scheduling Issues*

An efficient resource allocation strategy often requires accurate resource status information, which not only includes static attributes such as CPU speed, total memory, and other system-wise configurations but also time-varying properties with high dynamics such as current available CPU time, load average, remaining disk space, etc. In addition, it is imperative to ensure that the basic computing environment of a machine is well validated before injecting real job payloads in order to minimize unnecessary waste of computing resource. Empirical observations often indicate that a perfectly functioning compute resource can still fail to accomplish the task as a result of missing pieces of software, staging errors of input files and/or temporary unavailability of network connection, etc. However, the individual machine profile, being remote to the local administrative domain, is usually not directly obtainable; that is, the external resource often appears as a black box to the local user. Without reliable real-time status, it is difficult to apply scheduling policies accurately tailored for the job requirements idiosyncratic to different research and engineering projects.

B. High-level View of the Proposed Architecture

The workload management architecture under the PanDA framework achieves resource allocation by systematically sending a series of precursor jobs, namely pilots, to prepare candidate resources before fetching real payloads via PanDA server. Fig. 1 illustrates the PanDA system when applied to a typical Grid environment with multiple participating sites contributing resources. As shown from the figure, the user submits jobs to PanDA server via a simple client interface where each job defines the associated input and output files, desired matching criteria, and secure channels (e.g. HTTPS, GSIFTP, etc) from which job payloads can be obtained, etc. Note that the payload, in general, refers to any executables required for completing a task. These user jobs are then transmitted to the PanDA server via a secure HTTP, authenticated using Grid proxy certificates, followed by returns of submission status information to the client software. The pilot generator comes into the picture for disseminating pilots periodically to candidate compute resources with an adjustable rate. Optionally, PanDA server communicates with one or more distributed data management systems such as ATLAS DDM [13] to pre-stage input data required for given user jobs. Nonetheless, DDM and details of its data movements are complex subjects in their own rights and are not the primary focus of this paper.

As suggested in Fig. 1, PanDA architecture follows the separation-of-concerns principle [17] by decoupling job submission, job retrieval, data management, and resource allocation to distributed components. In this manner, the architecture has the advantage of higher adaptability; that is, users are free to choose locally-customized systems or preferable platforms for each role in the flow of workload processing, thereby achieving software interoperability as is highly preferable in general and complex Grid settings [16]. For instance, pilots can be distributed via Condor-G [5], a Condor system extension that allows for jobs to be submitted over the Grid through Globus-enabled gatekeepers that bridge between sites across administrative domains. Other contemporary batch systems such as Glite [24] and PBS [8] are possible alternatives for pilot submissions.

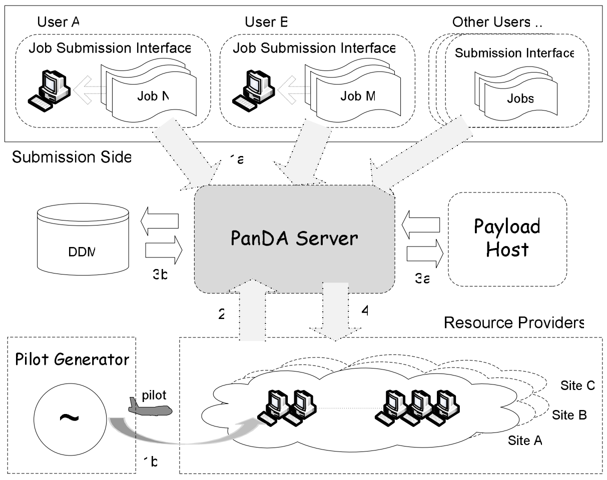

Fig. 1 High-level view of the PanDA architecture

Under PanDA architecture, efficient job dispatch and resource utilization hinges upon timely resource discovery, job sandboxing, and impromptu status reporting of target machines. To this end, a more scalable and automated version of pilot generator, i.e. *pilot factory*, is developed based on Condor's glidein mechanism [5].

The rest of the paper is organized as follows: Section 2 briefly introduces the design and current implementation of the PanDA system. Section 3 covers an overview of Condor [6], [7], [21] and discusses the role of Condor-based *pilot factory* in resource allocation. Numerical results that compare regular pilot dispatch using Condor-G and pilot-factory approach using Condor glidein is presented in section 4. Section 5 describes the experience with the PanDA system on active scientific experiments. Section 6 outlines related study and open issues in workload management. Lastly, implications of the PanDA-PF WMS and related future research are presented in Section 7.

II. PANDA AND PILOT GENERATOR

A. Terminology

Before proceeding to the details of the PanDA framework, it is helpful to elaborate some of the terminology used throughput the paper.

The current resource-sharing environment mainly includes two different categories: i) Grid Computing environment where the compute resource generally refers to the dedicated computing farms affiliated with certain organizations – such as companies, research labs, etc – that are mutually accountable, and ii) Volunteer Computing environment in which public processing and storage resources (typically PCs) are combined in an efficient way both architecturally and algorithmically as a conjoint effort to support computational needs from complex projects. In a futuristic sense, the second category can be extended to include any online computational devices such as mobile computers, sensor network, etc. Since the ultimate goal of the PanDA-PF WMS is to adapt to a general resource-sharing environment regardless the underlying middleware, the party that provides compute resources is generally referred to as a *resource provider*. This is analogous to the *site* affiliated with particular Virtual Organizations (or VOs) in the Grid literature.

In a computational cluster, the particular server that manages the backend compute resources is referred to as a *head node* (or a front-end node) to be generic; it is analogous to, in Globus architecture, the *gatekeeper* machine which typically has a batch system installed such as Condor or PBS in order to further schedule the jobs to the desired backend resources. Such backend resources are sometimes referred to as *worker nodes* (WNs) in Grid Computing context [14], [19].

For convenience, *resource boundary* is defined to refer to the set of available compute resources that are reachable from the local administrative domain. Resource boundary morphs as the external resources join or leave the local domain.

B. *Brief Overview of PanDA Architecture*

PanDA, Production ANd Distributed Analysis system, has been developed since Fall 2005 to support petabyte-scaled and data-driven production and distributed analysis processing in the ATLAS experiment [19]. The main focus here is to introduce the important features of PanDA directly linked to the architectural benefits as a general WMS and flexible, on-demand access to distributed resources through pilot mechanism.

To start with, there are four essential components in the PanDA system as illustrated in Fig. 2:

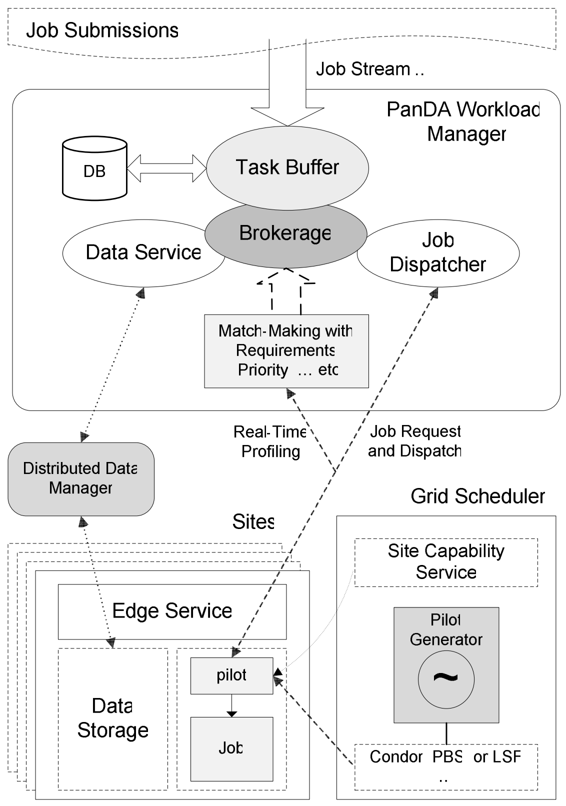

Fig. 2 Job stream flowing into the PanDA system as it interacts with the external data storage system to pre-stage the required dataset while pilots fetch appropriate jobs to their host resources

- *Task Buffer* represents a job repository containing user job information including related input and output files, various system requirements, job type, priority scheme, location of job’s payload, etc.
- *Data Service* interfaces with distributed data management system (e.g. ATLAS DDM) that performs stage-in and stage-out of the data on which the user job depends.
- *Job Brokerage* is a match-making component that prioritizes and assigns tasks on the basis of known static attributes such as job type, user-defined priority, locality of input data, required resource capacity (e.g. CPU speed, memory, disk space, etc), and other VO- or site-specific brokerage criteria. Job Brokerage in combination with pilots, distributed over candidate resources, completes the desired match-making cycle where pilots further provides dynamic, real-time resource attributes to help with the decision process of job assignments. Different scheduling policies can be supported by Job Brokerage, consistent with local site’s administrative requirements.
- *Job Dispatcher* follows the secure link specified in user jobs and sends job’s payloads to the designated compute resources upon pilot requests.

In a generic setting, PanDA Task Buffer functions as a system-wide, attribute-rich job database that records both static and dynamic information on all jobs submitted over the Grid. There is no inherent restriction on the task representation and client interface to the PanDA’s Task Buffer. In current implementation, PanDA uses a LAMP stack in which job submission is accomplished via a simple Python- and http-based client without dependency on the underlying Grid middleware. Job specifications are parsed and stored in PanDA’s backend database. Other possible job abstraction schemes for the PanDA front-end client include XML-based job specification, UDDI framework [20], and Condor classAd [7], etc.

While jobs are being uploaded to Task Buffer, the pilots in the meantime are running on the candidate resources, in the process of which, pilots then make requests to Job Dispatcher in order to obtain job payloads that match with pilot-resident hosts. Behind the scene, the best-fit job is determined by querying Job Brokerage that executes a given match-making algorithm. The decision process is based in part on the user job requirements and preferences and in part on the real-time snapshots that pilots had taken from their hosts. Allocation of jobs is followed by the dispatch of corresponding input data, handled by Data Service, to those pilot-resident hosts; during this process, Data Service interfaces with DDM, responsible for data movements, to obtain the desired data. In the PanDA framework, data pre-placement in target machines is ensured before the start of job execution to avoid failures from data staging, which usually in turn results in esoteric failure modes and waste of available computing resource such as CPU time. Data pre-staging is part of the schemes that implement the late-binding policy used in the PanDA framework as further discussed in the following subsection.

C. *Pilot*

Resource sharing mechanism in the current landscape of the Grid environment can be classified into two types: one is the push system where user tasks are allocated, or pushed, to the available resources via the task scheduler following certain match-making policy; conversely, the other type belongs to the pull system where resources, often volunteer hosts like personal computers, initiate connections to the designated task servers, pulling jobs to the hosts where scheduling policies are being executed from within volunteer's domain. Common batch systems used nowadays (e.g. Condor, PBS, LSF [10], etc) in the Grid community mostly falls into the push-system category. Volunteer computing systems such as BOINC [9] are mostly considered as pull systems – compute resources are often precluded behind firewalls or NATs and hence, resource sharing requires initiations from the machine's end.

The PanDA system, on the other hand, is a hybrid system following so called late-binding strategy in that user jobs are eventually bound to best-fit resources by first pushing pilots to the candidate resources to perform computing environment provisioning, followed by pulling actual job payloads through the PanDA server.

In the most generic use case, a pilot functions as a light-weight user job that validates the most rudimentary resource properties such as shell environment, interpreter/compiler availability, basic software stacks, system configurations and network connectivity and additionally, performs real-time resource profiling such as the remaining memory capacity, etc. These checks are performed to secure a basic working environment to the end-user and also provide a snapshot of various resource availability used to facilitate an optimal match-making process.

In theory, pilot can be designed hierarchically with a generic layer that performs only a high-level system-wise validation prior to its immediate binding with the site-specific service layers, which are implemented as a separate pilot core, containing site's local services such as methods for file transfer, security and sandboxing mechanisms, etc. Once the computing environment checks are completed, pilots then proceed to the following job-specific routines [14]:

1) Data Transfer: After receiving the job payload from PanDA server, the pilot invokes Data Service to draw in the required input data from DDM using site's preferable copy tools (e.g. GSIFTP); pilot also transfers output and log files back to end users upon completion of the designated task. Depending on the implementation scheme, file stage-out can also be performed instead by DDM in response to pilot requests, conforming more strictly to separation-of-concerns principle.
2) Job Execution: Pilot spawns a process as a job wrapper that copies input files, sets up runtime environment, executes the job payload, transfers job output files (either by itself or by delegating to DDM) and then finally performs final clean-ups. Conversely, if no job is received, the pilot simply cleans up the work directory and exits.
3) Monitoring: The pilot runs job monitor as a separate thread that tracks the runtime states and packs the information in terms of periodical heartbeat messages back to the Job Dispatcher at the PanDA server. If Job Dispatcher does not receive the message after a pre-defined period, it will consider that the job had failed and thus notify the pilot to kill the job. Moreover, each job's runtime information is updated accordingly upon the receipts of heartbeat messages.
4) Job Recovery: Temporary unavailability of the resource can often lead to job's valid outputs being stranded at remote storage system or worse yet, being deleted by clean-up operations from the resource provider itself. Site maintenance, system overhead, or job preemption enforced by site's local policy could all results in such temporary disconnect. The pilot in this scenario will attempt to rerun the entire file transfer mechanism mentioned earlier; if failed, the same file-transfer operation will then be executed by the successive pilots until a per-defined limit is reached.

The PanDA system accomplishes match-making process through not only the static job requirements stored in Task Buffer but also the dynamic resource attributes published from the pilots running at candidate machines. Under this modality, the resource utilization would highly depend on the way pilots are distributed and the functionality they offer. Apart from the generic pilot and hierarchical pilot discussed above, another possibility is to have pilot request multiple user jobs (hence, their payloads) simultaneously once the computing environment preparation is completed. Although such multi-tasking pilot is theoretically possible to realize, it often results in difficulty in maintaining fairness of resource share and could potentially lead to machine overload. For reasons above, achieving the optimal scheduling result is then being delegated to the pilot submission mechanism – the pilot generator.

D. *Pilot Generator*

In the PanDA framework, pilots are distributed to remote resources via an independent system tied to an underlying scheduler (e.g. Condor), as can be seen from both Fig. 1 and Fig. 2. An advantage of this scheme, particularly for the interactive analysis in research projects where minimal latency from job submission to launch is expected, is that the pilot dispatch mechanism bypasses any latency between pilot submission and execution – the user obtains, from the remote resource, an interactive session within a short duration so long as that there is at least one pilot, out of the population running over active resources, presents a valid computing environment for the job's payload at the time of need.

In this manner, the pilot mechanism isolates workload jobs from compute resources and batch system failure modes in that a workload job is assigned if and only if the pilot successfully launches on a candidate resource. Throughput of user jobs is increased since there is a continuously ongoing service of resource provisioning from distributed pilots running in parallel as jobs are submitted. As a result, machines effectively appear available on-demand for end users. In addition, the pilot service layer isolates the PanDA system from Grid heterogeneities, being encapsulated in the pilot, such that from the perspective of end users, the Grid or the resource-sharing environment in general appears homogeneous.

The PanDA framework places no restrictions on the mechanism by which pilots are disseminated. In fact, this highly depends on resource provider's preferable batch system, which is part of the heterogeneous factor on resource-sharing landscape. Where PanDA was originally introduced, the US ATLAS production has been primarily using Condor-G to schedule pilots across site boundaries [19]. Yet, Condor-G encounters scalability issues at Globus-controlled gatekeepers as a result of high GRAM traffic in response to high-volume of user jobs. GRAM [26] refers to the Grid Resource Allocation and Management protocol that supports the submission of remote job requests and their subsequent monitoring and control. With an increase in user job demands, often seen in complex projects (e.g. ATLAS experiment), a higher pilot flow is expected accordingly, which in turn leads to heavier GRAM traffic (i.e. channel between Condor and Globus software). To achieve a more scalable pilot dissemination, a distributed scheduling approach based on Condor's glidein mechanism is therefore conceived. Details are to be covered in the next section following a brief overview of the Condor system.

III. Condor and Pilot Factory

Condor [7] is a distributed workload management system developed primarily for integrating distributed resources to ultimately achieve both high-throughput computing [22] and opportunistic computing. Similar to other batch systems, Condor provides the following major functionality: job/machine monitoring and management, fault recovery, checkpointing, customizable scheduling polices and match-making mechanisms that reflect job/machine requirements and different priority schemes [23]. Fig. 3 presents the fundamental structure of Condor system. The core logical components, also known as Condor kernel [7], include job queue (functionally represented by the Condor schedd daemon), virtual machine (startd daemon), match maker (negotiator daemon), and in-memory database (collector daemon).

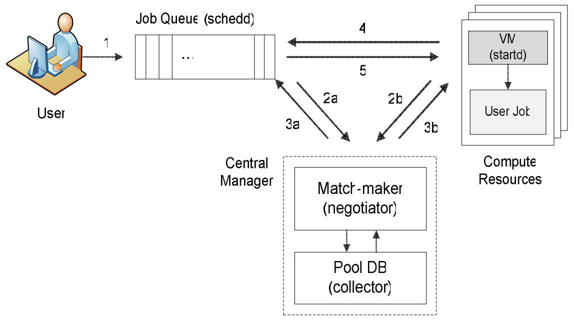

Fig. 3 Condor kernel

The following stepwise description briefly outlines how Condor works using the aforementioned components: i) users submit tasks to the job queue (i.e. schedd) in the format of ClassAds containing matching criteria ii) the schedd publishes all task information to the pool database (i.e. collector) while, in the mean time, Condor virtual machines (i.e. startd), being distributed over all the available compute resources, also advertise their associated machine profiles to the collector including system configurations, machine runtime states, and matching preferences (over user jobs), etc iii) with the collector receiving information from both the job queue and resources, the Condor negotiator then executes its match-making algorithm based on the scheduling policies defined in both the job and the resource (in terms of requirements and preferences) and finally determines the best match.

A. Condor Glidein

A basic Condor-managed resource pool consists of the following building blocks: i) the *job submitter* with one or more job queues (i.e. schedds) containing submitted user jobs, ii) the *job executor* consisting of one or more distributed virtual machines (i.e. startds) that represent all the available compute resources, and iii) the *central manager* (i.e. collector and negotiator) primarily responsible for collecting pool-wise status information and performing match-making algorithm. Each role mentioned above runs independently and can be deployed on different machines. Condor system, being structurally decentralized in its design, makes it possible to dynamically deploy partial Condor functionality on-demand (i.e. subset of Condor daemons) across the network, whereby expands the local resource on the fly. The idea of dynamic deployment gives rise to *glidein*. A Condor glidein generally refers to the startd and its functionally-dependent daemons – altogether serving as a virtual machine – that are dynamically installed and executed on a remote resource. Glidein startd creates an abstraction of the hosting machine in terms of the Condor representation and advertises itself to the local-pool database (collector) such that the remote resource effectively joins the local pool and become visible to the local user.

B. Schedd Glidein

The schedd-based glidein, similar to the dynamically-deployed Condor virtual machines mentioned previously, is accomplished by remotely install and execute a subset of Condor daemons altogether functioning as a job queue. This remote schedd effectively "glides into" the local resource pool by advertising itself to the local collector, sharing exactly the same mechanism as the startd glideins.

Fig. 4 illustrates the Condor glidein mechanism and compares the two different glidein types with one working effectively as distributed virtual-machines and the other as a dynamic job queue. Typically, the schedd glidein is deployed on a remote Globus-enabled machine where the glidein operates as a medium that redirects Condor jobs to the site's native batch system. Contrary to the job flow in the Globus model, user jobs now flow through the schedd glidein to the remote batch system rather than through the Globus Job Manager, a set of processes that perform monitoring and control over Grid jobs. The downstream flow remains the same as the Globus model where the remote batch system is responsible for eventual match-making for the incoming jobs.

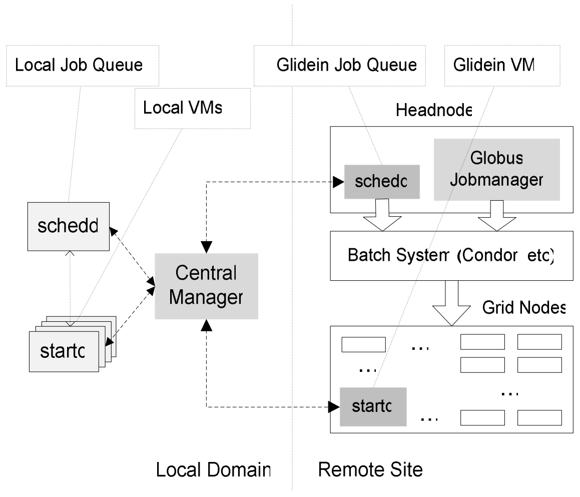

Fig. 4 Glidein schematic

The glidein schedd communicates with the native batch system through the related GAHP server process depending on the native batch system type (e.g. Condor, PBS, LSF, etc). For the purpose of presenting the glidein-based approach for pilot dispatch, here it is assumed that the remote batch system type is Condor, in which case the glidein relays jobs using the Condor-C mechanism [25]. Condor-C stands for Condor to Condor, designed to interconnect two or more independent Condor-managed resource pools. Using Condor-C, jobs are submitted to a job queue (i.e. client schedd) and are subsequently forwarded to another job queue (i.e. server schedd). The client schedd is typically an instance running on a machine within the local resource domain (i.e. the local job submitter) whereas the server schedd runs in the foreign resource domain. When the server schedd is the glidein instance, remote jobs can effectively be allotted to a foreign computational cluster while being treated almost the same as local jobs. In this manner, multiple clusters across administrative domains are virtually merged together, thereby expanding the resource boundary.

The glidein schedd works similarly to the role of the Globus Job Manager in the sense that the glidein also serves as a resource broker connecting different resource domains. The fundamental difference is that the jobs dispatched via a glidein schedd no longer go through the Globus GRAM channel. Compared to the Condor-G using GRAM protocol, each job, either in active or wait state, is monitored and controlled by a Globus *jobmanger* process (a primary component of the Globus Job Manager), leading to higher resource consumption upon heavier job flow. In the Condor-G model, such overhead due to job monitoring activities is ameliorated by introducing a Grid Monitor that temporarily shuts down the *jobmanager* process while the associated job is not running [25]. However, the source of the overhead still exists due to the remaining monitoring activities in active jobs, which are required considering that each user job is unique and that the job representations are inherently different in the Condor and Globus system[1].

While Condor-G fits the needs of regular user jobs, it may be excessive for the pilot mechanism since pilots in aggregate work as a light-weight service layer on top of the job payloads. The schedd glidein can thus achieve higher scalability for the pilot mechanism by treating pilot jobs as a "homogenous job stream," requiring no separate job monitoring and control. In addition, a glidein by definition is only deployed on a service-on-demand basis and thus can be removed when no jobs are intended to use the resource pool. The next section introduces an application of the schedd glidein used in pilot disseminations.

C. *Pilot Factory*

The Pilot Factory (PF) represents an independent and automatic system for the pilot dispatch and control. It is developed in parallel with PanDA system and for historical reasons, Pilot Factory took its name to differentiate it from a regular pilot generator (or pilot submitter) used only as a component in the factory. The factory first deploys glidein schedds to the head nodes of the sites, followed by the backend pilot generator submitting pilots directly to these glideins, from which these pilots are then redirected to the native batch system.

The Pilot Factory consists of three major components: i) a glidein launcher, responsible for the dynamic deployment of glideins to eligible sites ii) a glidein monitor that detects any failure or removal of the running glideins due to walltime limits or temporary site downtimes, upon which the monitor then invokes the glidein launcher to deploy new glidein instances and iii) a pilot generator that distributes pilots through the schedd glideins running on remote resources. The core of the glidein launcher and monitor lies in the mechanism to submit glidein requests, which is accomplished by initiating Condor-G jobs to configure, install and execute related daemon set on the target head node of the foreign site. The pilot generator is built upon Condor schedd, to which pilots are submitted. Given that all factory components are essentially complex wrappers over Condor, they can be distributed, like the majority of Condor daemons, to different machines without locality constraint.

The current implementation of schedd glidein still relies upon the service of Globus software for its initial setup in that the glidein deployment is achieved via two consecutive Condor-G jobs: the setup and the startup. The system-setup job locates and installs platform-dependent, schedd-related binaries on the designated head node, generates the required configuration file and a startup script to be used at the next phase. The startup job then executes the script staged earlier to activate Condor daemons. Using the glidein as a job queue for pilots, Globus software only serves occasional glidein requests

[1] Condor-G converts the job description to RSL (Resource Allocation Language) format used by GRAM. The Globus Job Manger then parses the RSL that specifies the binary to be executed and other job requirements such as CPU time, number of processors, etc, some of which are further used to construct the job submit file for the native batch system.

and thus is fully decoupled from the pilot traffic for as long as the glidein remains active.

As alluded to earlier, Condor schedd supports the interfacing with multiple widely-used batch systems (e.g. PBS) in addition to Condor itself with the proper configuration and the required GAHP server binary. Using the feature above, the schedd glidein mirrors the external resource domain, thereby hiding the heterogeneity of site infrastructure. The result effectively presents the pilot generator with a uniform submission portal yet without the burden of constant job control/monitoring as in Condor-G, which allows for much higher pilot flow. The Pilot Factory approach therefore has great potential to lift the performance bottleneck in the PanDA architecture in light of its higher scalability and its flexibility in the on-demand deployment. The next section justifies this modality with empirical results.

IV. Experimental Results and Analysis

Since PanDA was introduced, the US ATLAS production has been primarily using Condor-G to schedule pilots across site boundaries. At times of peak usage, large pilot traffic is often required to cope with high demand of user jobs. To alleviate the correspondingly high GRAM traffic in Condor-G-based pilot dispatch, the Pilot Factory approach is developed as an alternative method that substantially reduces the need of Globus brokerage by deploying Condor's schedd glideins to the front-end nodes of the remote clusters. Pilots are then flow through the glideins to the native scheduling system where glideins function as tunnels that connect the pilot submitter host with the remote scheduler.

A. *Resource Usage Comparison*

In this section, an experiment is presented to compare the resource usage in Condor-G-based pilot dispatch with the Pilot Factory approach in terms of percentage CPU time and memory consumption. The experiment was conducted in a test computational cluster (OSG-ITB test bed) with 1 head node and 8 worker nodes (i.e. backend compute resources), configured with 8 job slots; that is, a maximum of 8 jobs is allowed to be in the running state on the job queue.

To ensure a continuous supply of pilot jobs, the pilot generator was configured to maintain a queue depth of 20 pilots on the submitter host so that ultimately, the 8 job slots on the remote cluster are filled most of the time; although theoretically, a persistence of approximately 8 pilot jobs (or less) should suffice. To simulate the fact that pilots in practice should remain active for as long as their associated user jobs of varying execution times, each pilot is configured to have a runtime determined by the bounded Gaussian distribution with an appropriate lower and upper limit (see Table I). Further, the resource usage metrics (e.g. percentage CPU time) on the cluster-head node are sampled on a predefined interval perturbed by a Gaussian noise. The irregular sampling interval is incorporated here with the intent of minimizing biased measurements from any possible "synchronization" between hidden temporal patterns in the scheduling process and the sampling process itself. The specification for the experimentation is summarized in Table I.

TABLE I
Pilot-Specific Parameters

Parameter	Value
Number of job slots in the target cluster	8
Pilot queue depth	20
Pilot runtime	Minimum: 5 seconds Maximum: 180 seconds
Sampling rate of resource usage	15 samples per miniute

.

Fig. 5 compares the resource usage in terms of percentage CPU time while Fig. 6 compares memory usage. With the same pilot load, the result indicates that the Pilot Factory mode has lower resource consumption on average and lower sampling errors. Since a glidein only serves as a conduit between the submitter host and the remote scheduler, computing resource is only allocated for running the schedd and the GAHP server process without additional processing time required for the per-job monitoring/control as in the case of Condor-G.

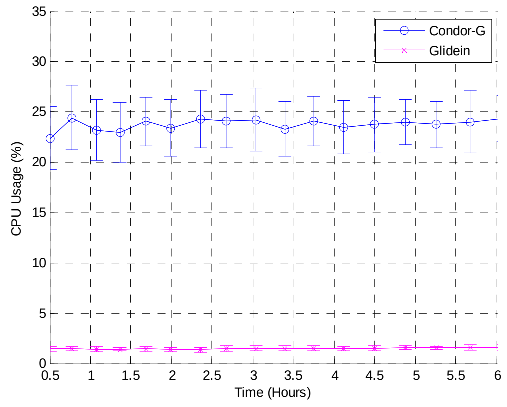

Fig. 5 Percentage CPU times during a 6-hour time frame

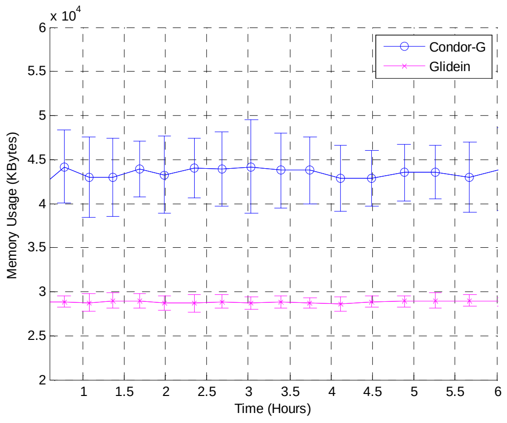

Fig. 6 Memory usage during a 6-hour time frame

Note that the experiment focuses on relatively shorter jobs with life spans within the order of a few minutes (3 minutes at maximum in this experiment). Theoretically speaking, the shorter the jobs, the higher the overhead in the case of Condor-G since each job requires a *jobmanager* process being active during the phase of job submission, file staging, and the cleanup upon job completion. Consequently, as long as shorter jobs are in the majority, the resource usage with higher workload is expected to be similar to (if not higher than) the empirical results presented here. Lastly, since a *jobmanager* and its child processes appear at various stages of a Condor-G job prior to its completion, both the CPU and memory usage tend to fluctuate more than those in the PF submission mode, resulting in higher sampling errors.

V. Experience

The development of the PanDA system dates back to the late 2005 to meet ATLAS requirements for efficient processing and management of large-scaled production tasks and distributed scientific analysis. The PanDA architecture gives rise to a dynamic workload management system that optimizes resource utilization through data-driven scheduling and just-in-time resource allocation with the pilot mechanism. The benefits of PanDA's architecture have led to its widening use in OSG [27] and EGEE [18], etc. In particular, PanDA has processed more than 70 million jobs as of late 2009, currently at a typical rate of about 1M jobs per week for production at approximately 120 sites around the world, and about 20K jobs per day for distributed analysis. In view of the potentiality of combining broader forms of compute resources across geographic boundaries, the PanDA architecture is further extended to a more generic framework that adapts to the heterogeneity of Grid infrastructures, thereby providing a uniform job-management service layer and achieving optimization of resource allocations with a late-binding strategy as emphasized in this paper. These efforts lead to a collaborative research with the Condor team and the development of applications in *glidein* technology such as the Pilot Factory as described in Section 3.

VI. Related work

The late-binding strategy used in PanDA for resource allocation is realized in two contexts: data-driven scheduling, and just-in-time match-making. In the PanDA framework, both the aforementioned services are in part delegated to the distributed pilots at the candidate resources in the sense that pilots can be configured to initiate data movements and collect real-time resource profile for match-making purposes. The concept of the pilot mechanism, in its late-binding with data, can be traced back to the experience in DIRAC [28] used for the LHCb experiment. DIRAC provides several architecture-level solutions for reliable data distribution, data integrity and access in order to minimize waste of resource due to failure modes during data staging process. Once the required sets of data become available and are validated, the workload agent in DIRAC then submits to the Grid the jobs that have been waiting for these data.

From the prospective of the late-binding between jobs and resources, the pilot identity is analogous to the role of remote client agents (or workers) in Volunteer Computing systems such as BOINC [9], SETI@home [4], and Distributed.net [29]. In these systems, each volunteer host is attached to the servers from which tasks can be downloaded. Various CPU scheduling schemes on the level of work-fetch policy, CPU time-slicing, estimate of completion time, etc, are then enforced by the client agent running on the volunteer host [30]. The acquired tasks, as a result, can be tightly matched with real-time machine properties of the volunteer host. Such a client-server model also exists in the form of the pilot mechanism in the PanDA architecture; yet the pilot approach works slightly differently in that there is no pre-defined agreement that associates user tasks with the target machines where pilots are injected. Consequently, the real-time resource information collected by pilot jobs is sent back to the PanDA server where scheduling strategy is dynamically determined by selecting the best-fit user task mutually agreeable to the target resource. In this manner, scheduling algorithms are then decoupled from the client agent (i.e. the pilot) that initiates requests for user jobs.

Efficient cross-domain resource allocation is a key aspect for minimizing heterogeneity of the resource-sharing environment. This subject has been addressed by many related research including Condor-G, BDII [31], and other edge services such as MDS (Monitoring and Discovery Service) from the Globus project [11]. Condor-G now incorporates a site-level resource allocation mechanism by which user jobs are matched to the desirable sites without having to explicitly select the target site. Grid resources identify themselves by advertising their available services, requirements and preferences over jobs in the form of ClassAds while user jobs also specify the likes; a match occurs when the overall requirements between a job and a Grid resource are compatible with each other. However, achieving this high-level brokerage requires the sites to cooperate by providing consistent and pre-defined resource descriptions that accurately reflect the capacity of their managed resources. BDII, on the other hand, periodically polls resource attributes, such as free CPUs, supported Virtual Organizations, etc, through LDAP servers gathering information from computational clusters. However, using the BDII approach for resource allocation still requires an agreement and consistency over the resource profiles from their providers. In addition, the real-time resource information is obtained through constant polling (e.g. using periodical cron jobs) to the related servers in the target site domain. This architecture, when compared to the pilot mechanism, would require dedicated servers per site and thus, may not generalize as well to the general resource-sharing environment such as the network of volunteer hosts.

VII. Conclusion and future work

Abstraction of the resource-sharing infrastructure for a homogeneous representation has been one of the primary goals in the Grid computing community. The PanDA-PF architecture is presented here to achieve a uniform view of the Grid and better resource utilization through the layer of distributed pilots and their efficient dispatch. The pilot mechanism accelerates distant resource discovery and, through late-binding between tasks and their target machines, minimizes computing resource wasted in various failure modes. Concurrently, pilots gather real-time resource properties so as to make achievable a seamless match-making procedure. Heterogeneity of the Grid is encapsulated in the layer of the distributed pilots so that users do not have to deal with the differences in the underlying schedulers and other specifics in the fabric layer [16] of the Grid. Further, the PanDA-PF architecture also aims to generalize resource utilization to a broader form of distributed resources such as volunteering computing nodes across the Internet where pilots can act as client agents that initiate requests for user jobs.

The Pilot Factory is a glidein-based solution to a more scalable pilot dispatch than the conventional Grid-job approach. In the Pilot Factory mode, the glideins as dynamic job queues are first installed at the target cluster head nodes prior to pilot dispatch. This is mainly accomplished through appropriately configured Condor-G jobs using Condor schedd and its related daemons as executables. Subsequently, the light-weight repetitive jobs such as pilots will then flow through glideins to the native scheduler without excessive monitoring/control on the per-job basis. Treating each pilot individually as a Grid job often leads to scalability issues due to the correspondingly large GRAM traffic, which is required in pilot disseminations to cope with large and constant job flow. In particular, heavy workload is often expected during the course of large-scale and complex scientific projects such as the ATLAS experiment.

The PanDA-PF WMS provides users with an architectural foundation for resource acquisition, validation and allocation with user jobs. However, there are other dimensions in harnessing distributed resources not yet fully investigated within the PanDA framework such as the following: i) strategic expansion of resource boundary (i.e. increasing the set of available resources) by distributing Condor glideins (or VMs in general) on compute resources fitted for user jobs, and ii) adaptive match-making policy that improves itself through learning the dynamics of jobs and computing resources.

In support of the preceding objectives, glideinWMS [32] can be used to expand the resource boundary by skillfully distributing Condor glideins at the target computational clusters as locally-accessible virtual machines. Furthermore, the decision-making process that integrates reinforcement learning with the cluster-based conceptual model [33] provides an initial effort towards solving the match-making optimization problem characterized by morphing resource boundary formed by distributed pilots or glideins. These techniques could potentially increase the productivity of the PanDA-PF WMS by several magnitudes through a strategic pilot dispatch with the Pilot Factory in addition to the optimal resource allocation within the PanDA framework.

ACKNOWLEDGMENT

This work was completed using resources provided by the Open Science Grid, which is supported by the Office of Science, U.S. Department of Energy, SciDAC program under Contract DE-FC02-06ER41436 and the National Science Foundation Cooperative Agreement, PHY-0621704. We also thank the PAS and RACF Groups at the Physics Department in Brookhaven National Laboratory for their feedback, guidance and assistance in the required software systems and computing facilities.

REFERENCES

[1] K. Harrison, R.W.L. Jones, D.Liko, C.L. Tan, "Distributed Analysis in the ATLAS Experiment," in Proc. AHM Conf., 2006.
[2] S. Kolos et al., "Online Monitoring software framework in the ATLAS experiment", CHEP 2003, La Jolla, California, USA, 2003.
[3] Akihiko Konagaya, "The Grid as a 'Ba' for Biomedical Knowledge Creation," Grid Computing in Life Science, LSGRID 2005, pp. 1-10.
[4] W. T. Sullivan, III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, D. Anderson. A New Major SETI Project Based on Project SERENDIP Data and 100,000 Personal Computers. Astronomical and Biochemical Origins and the Search for Life in the Universe, Proc. of the Fifth Intl. Conf. on Bioastronomy. 1997.
[5] J. Frey, T. Tannenbaum, M. Livny, "Condor-G: A Computation Management Agent for Multi-Institutional Grid", Cluster Computing, Springer Netherlands, 2004, pp. 237-246.
[6] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Inc., 2002.
[7] T. T. Douglas Thain and M. Livny. Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, 2004.
[8] Papakhian, M. Comparing Job-Management Systems: The User's Perspective. IEEE Computational Science & Engineering, (April-June) 1998. Available: http://pbs.mrj.com
[9] D.P. Anderson. "BOINC: A System for Public-Resource Computing and Storage," 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, 2004, pp. 365-372.
[10] Zhou, S. LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems. Proceedings of the Workshop on Cluster Computing, 1992.
[11] Foster, I. and Kesselman, C. The Globus Project: A Status Report. In Proc. Heterogeneous Computing Workshop, IEEE Press, 1998, pp. 4-18.
[12] P.Nilsson, J.Caballero, K.De, T. Maeno, M.Potekhin and T.Wenaus, "The PanDA system in the ATLAS experiment," ACAT 2008 Conference Proceedings.
[13] Klimentov A., "ATLAS Distributed Data Management Operations. Experience and Projection," Journal of Physics: Conf. Series, 2007.
[14] Nilsson P., "Experience from a Pilot based system for ATLAS, " Journal of Physics: Conference Series, 2008
[15] M. Avvenuti, P. Corsini, P. Masci, A. Vecchio, "Opportunistic Computing for Wireless Sensor Network," IEEE Intl Conf. on Mobile Adhoc and Sensor Systems," 2007, pp. 1-6
[16] Foster, I., Kesselman, C., and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Intl. J. Supercomputer Applications, 2001
[17] B. DeWin, F. Piessens, W. Joosen, T. Verhanneman, "On The Importance of the Separation-Of-Concerns Principle in Secure Software Engineering," In ACSA Workshop on the Application of Engineering Principles to System Security Design, 2003, pp. 1-10.
[18] Enabling Grids for E-science. Available: www.eu-egee.org
[19] T Maeno, "PanDA: Distributed Production and Distributed Analysis System for ATLAS," Journal of Physics: Conference Series, 2008.
[20] Organization for the Advancement of Structured Information Standards, "Introduction to UDDI: Important Features and Functional Concepts," 2004.
[21] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In Proc. 8th Intl Conf. on Distributed Computing Systems, 1988, pp.104-111.
[22] Jim Basney, Miron Livny, and Todd Tannenbaum, "High Throughput Computing with Condor," HPCU news, Volume 1(2), June 1997.
[23] Rajesh Raman, Miron Livny, and Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," Proc. of the 7th IEEE International. Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL
[24] gLite, Lightweight Middleware for Grid Computing. Available: http://glite.web.cern.ch/glite/

[25] Condor manual, development release version 7.0. Available: http://www.cs.wisc.edu/condor/manual/

[26] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[27] Open Science Grid. http://www.opensciencegrid.org

[28] A. Tsaregorodtsev, V. Garonne, I. Stokes-Rees, "DIRAC: A Scalable Lightweight Architecture for High Throughput Computing," Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), 2004, pp.19-25.

[29] Distributed.net: The First General-Purpose Distributed Computing Project. Available: http://www.distributed.net

[30] Derrick Kondo, David P. Anderson and John McLeod VII. "Performance Evaluation of Scheduling Policies for Volunteer Computing," 3rd IEEE International Conference on e-Science and Grid Computing. Bangalore, India, December 10-13, 2007.

[31] CERN Twiki. http://twiki.cern.ch/twiki/bin/view/EGEE/BDII

[32] Igor Sfiligoi. *Structural Overview of the GlideinWMS*. Available: http://www.uscms.org/SoftwareComputing/Grid/WMS/glideinWMS/

[33] Chiu P, Huber M, "Clustering Similar Actions in Sequential Decision Processes," in Proc. of the 8th Intl Conf. on Machine Learning and Applications (ICMLA'09), Miami Beach, FL. 2009, pp. 776-781.