Experience from a pilot based system for ATLAS

Paul Nilsson on behalf of the PanDA team and ATLAS collaboration

University of Texas at Arlington, Science Hall, PO Box 19059, Arlington, TX 76019

Paul.Nilsson@cern.ch

Abstract. The PanDA software provides a highly performing distributed production and distributed analysis system. It is the first system in the ATLAS experiment [1] to use a pilot based late job delivery technique. This paper describes the architecture of the pilot system used in PanDA. Unique features have been implemented for high reliability automation in a distributed environment. Performance of PanDA is analyzed from one and a half years of experience of performing distributed computing on the Open Science Grid (OSG) infrastructure. Experience with pilot delivery mechanism using Condor-G [2], and a glide-in factory developed under OSG will be described.

1. Introduction to the PanDA system

The PanDA system is the ATLAS Production and Distributed Analysis system for the OSG [3][4] since September 2006 in an OSG joint project with the Condor project. It was developed by US ATLAS to meet the requirements for full scale production and distributed analysis processing for the ATLAS Experiment which is currently being constructed at CERN. ATLAS places challenging requirements on throughput, scalability, robustness, and operations manpower which demands an efficient integrated data/processing management. During full scale production, current estimates are for 100-200k jobs/day within US ATLAS, and about 4 times that number ATLAS-wide. PanDA development began in August 2005 and it took over US ATLAS production responsibilities in December 2005. Recently, Canadian EGEE sites have also adopted PanDA for their production.

1.1. PanDA features

PanDA was designed to support both managed production and user analysis via flexible job specification and injection. The dataset based organization of PanDA matches the Distributed Data Management (DDM) system and the analysis work model which is based on the ATLAS data management system Don Quixote 2 (DQ2) [5]. The asynchronous DDM system is used to pre-stage all input data and immediately return job outputs for production jobs. This minimizes data transport latencies and delivers the earliest possible results. The management and optimization of the workload via a job queue with a late binding of jobs to the worker nodes gives a dynamic and flexible system response to the highly variable Distributed Analysis (DA) work.

The use of grid and farm batch queues to pre-stage pilot jobs (job wrappers) to the worker nodes and directly deliver workloads from PanDA allows for a fast injection of DA work.

2. The pilot system

The PanDA pilot is a lightweight execution environment implemented in Python that is used to prepare the Computing Element (CE), request the actual payload from the PanDA server, execute the payload, and clean up. It handles data stage-ins and stage-outs between the worker node and the local Storage Element (SE). When the pilot is executed, all input data is already copied to the site by the DDM system.

The pilot jobs are broadcasted from the Job Schedulers to the batch systems and the grid sites. The actual payload is scheduled only when a CPU becomes available which leads to low latency.

2.1. Schedulers

PanDA is using a multiple approach to the pilot job submission. Primarily, the Condor-G system is used for US ATLAS production at most of the US production sites. In the PanDA job scheduler, a program called pusher is broadcasting pilots to all the available Computing Elements (CE). Information about the CEs is stored in a special configuration file and the scheduling algorithm is selectable [6].

A local batch scheduler has also been developed which is used with PBS [7] at UTA, TRIUMF and UBC, as well as with Condor at BNL, AGLT2 and TORONTO. The local job submitter is very efficient and robust, although it is not centrally managed.

A new generic scheduler, known as the "AutoPilot", is currently under development and testing. The scheduler supports local batch, Condor-G, LCG, and soon also pilot factories based on new Condor scheduler glide-ins [8]. It extends Condor-G support OSG- and EGEE-wide.

PanDA is running dedicated submit hosts at BNL, UTA, Madison and at Lyon. Furthermore, a CERN host for PanDA testing is coming online shortly.

2.2. Pilot features

The most notable pilot features are listed below:

- **Data transfer** of input/output and log files to/from local SE
  Pilots can be centrally configured according to each site policy. Until recently, the pilot consulted a static file containing the necessary site information needed for the file transfer: the copy tool and the location of its setup file. PanDA is however currently migrating to using a more flexible configuration DB that is replacing the static site configuration file. Special wrapper classes, here referred to as site movers, have been written for a variety of copy tools (*cp, dccp, rfcp, uberftp, gridftp, lcg-cp*). Direct reading support exists for dCache, Castor and Xrootd. A practical feature is the time-out control for hanging transfers. In case this happens, the pilot will abort the copy. The pilot allows for multiple attempts of the file transfer (site configurable).

- **Job execution**
  If the pilot does not receive a payload from the Job Dispatcher, it will clean up the temporary work directory and terminate. In the case it does receive a job, it will fork the job wrapper, known as the runJob script. This script copies the input files, sets up the runtime environment, executes the real payload and copies the output files after successful completion of the job. The pilot cleans up the work directory after the job is done (finished or failed) and saves a tar ball of the work directory to the local SE.

- **Monitoring**
  The job wrapper communicates with the pilot through TCP messages. The pilot runs a job monitor as a separate thread which checks whether the payload is still alive and is listening for any TCP messages. The job monitor sends heartbeat messages back to the job dispatcher

every 30 minutes. If the job dispatcher stops receiving heartbeats from the pilot job after six hours it will fail the job. The heartbeat message is recognized by the server with a receipt that can be used to send messages back to the pilot. Through this communication channel, a user can tell the pilot to kill the job. Every server update is followed by an update of a job state file which contains the current state of the job. The job state file is used by the job recovery mechanism described below.

- **Space report**
  At the beginning of the pilot execution, the pilot checks the available SE space and reports it back to the PanDA server. When the pilot is requesting a job from the dispatcher, it provides it with a report of available worker node disk space, memory and CPU.

- **Job recovery**
  The job recovery refers to the ability to recover lost jobs run by earlier pilots, and is discussed in the following section.

2.3. Job recovery

When there are site or service problems, valid data from completed jobs can be stranded on the worker nodes or even be deleted by cleanup scripts running at the site. The pilot is equipped with a recovery mechanism that attempts to salvage these lost jobs, without a rerun of the original job being necessary. If a job is unable to transfer its output to the SE, it leaves the output files and status on the worker node. A later pilot running on the same node checks for such stranded jobs, and retries the SE data movement if necessary after a minimum of one hour. If it fails again, it will leave the files for yet another pilot. Pilots will try a maximum of 15 times before failing the job permanently. The PanDA server can also fail the job if the recovery process has not finished within three days. A later pilot trying to recover the already failed job will be notified by the server to give up. An un-recovered job will be left on the worker node for manual inspection. Special cleanup scripts running on the worker nodes will remove the job traces after a few days, depending on the site.

The job recovery mechanism, which is optional, has proved to be highly effective in waiting out service disruptions for later recovery. CPU cycles otherwise lost are thus recovered.

2.4. Pilot workflow

When a pilot job is launched, it starts with scanning the local disk for remains of lost jobs (Fig. 1). In case it finds one or several such jobs, it will reattempt the file transfer and registration. The pilot then asks the job dispatcher for the real payload. If no job is available, it will quietly cleanup the work directory and exit. Otherwise, the pilot will fork the job wrapper and start to monitor the payload. The job wrapper copies all input files from the SE to the worker node disk and sets up the runtime environment before executing the payload. All steps are communicated via TCP messages back to the pilot monitor thread. The main pilot loop keeps track of the payload by checking that it keeps writing to its output files. If no update has been made in ten hours, it will consider the job to be failed. Every 30 minutes the pilot monitor sends a heartbeat message back to the job dispatcher. If the dispatcher does not receive any heartbeats for six hours, it will fail the job. The most frequent reason for lost heartbeats is a site problem where the work node was rebooted but can also be due to pilot problems. A crashed pilot can in some benign cases still be recovered by a later pilot.

When the payload finishes, the job wrapper transfers the output to the SE and sends a final message back to the pilot. Upon receipt of this final message, the pilot wraps up the work directory of the job and transfers the tar ball to the SE and sends a final update to the job dispatcher.

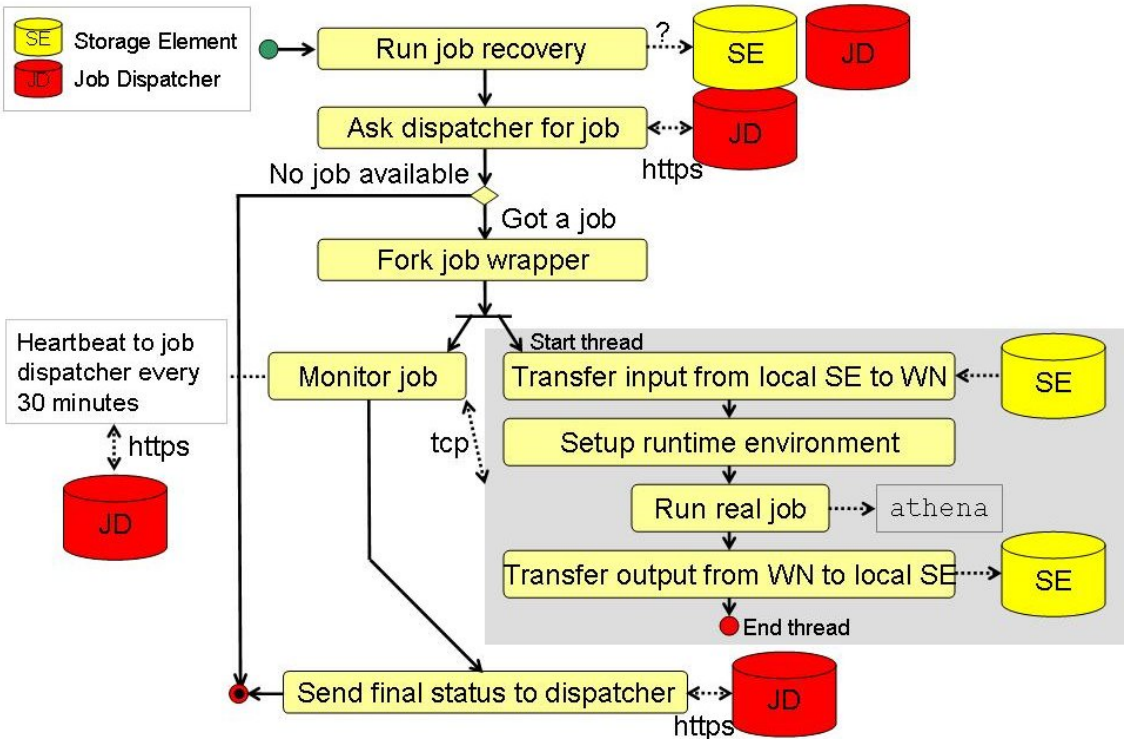
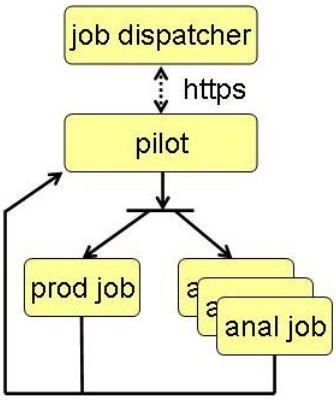


Fig 1. Pilot workflow.

2.5. Multitasking pilot

The pilot can be run in multitasking mode, which refers to the ability to execute a production job at the same time as a user analysis job by the same pilot (Fig. 2). In this mode, the pilot asks for new analysis jobs one after another until the production job finishes. A user analysis job is typically shorter than a production job. The multitasking pilot can offer a large pool of analysis resources, namely the production farm, without machines standing idle. This can be good for accommodating highly fluctuating analysis loads with very short latency, always ready to receive distributed analysis jobs.



However, it also raises some concerns such as resource contention (especially with memory), conflict with resource usage policies, and can break fair sharing of local batch systems with other users' jobs.

The multitasking pilot is pending evaluation. In the meantime, a traditional approach with special queues (long and short) for analysis jobs is used.

Fig. 2. The multitasking pilot executing a user analysis job, one after the other, at the same time as a production job.

3. PanDA performance

3.1. Efficiency

An automated error analysis mechanism is critical for scalability. As of September 2007, PanDA is serving ~10k jobs per day with a steady increase as new sites are coming online. A few percent of unknown failures will require the debugging of hundreds of log files per day, which is the most time consuming part of operations. Scaling to 100k+ jobs per day will be a challenge. To alleviate this problem, PanDA is relying on logging, monitoring and error reporting services:

- Logging: Independent https based logging service which allows PanDA components to log incidents in a database via the Python logging module
- Monitoring: Integrated web based monitoring system that provides an interface to PanDA for operators and users [9]
- Error reporting: PanDA reports 60+ different errors (and growing with experience)

PanDA system errors so far corresponds to less than 3% whereas the majority of the errors are site problems, or from other software systems (Fig. 3).

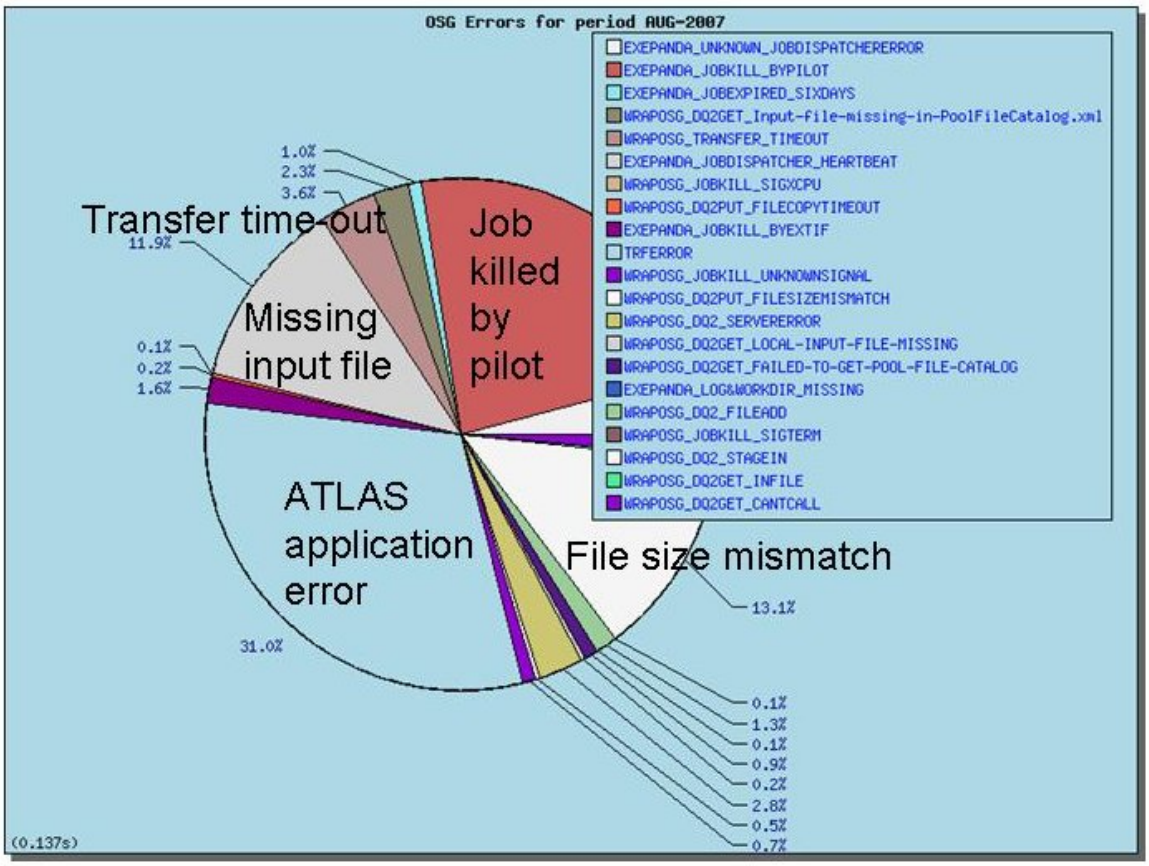


Fig. 3. Reported errors during August 2007.

Any found PanDA system error is usually fixed immediately. To date, no PanDA scaling problem has been seen with a maximum of ~40k jobs in the job queue.

4. PanDA and Condor glide-ins

Using Condor glide-ins has been planned for PanDA since October 2005 and has been actively pursued since September 2006. The initial target is to add schedd glide-ins to support a site-level pilot factory, and thus move away pilot submission from a global submission point as we have now, which will be a new capability for Condor. Presently, Condor only supports startd ('pilot' type) glide-ins. For this venture, PanDA is working directly with the Condor team and are collaborating with CMS on startd glide-ins. PanDA can easily use Condor startd glide-in pools to submit jobs.

5. Conclusions

PanDA production on OSG and EGEE is going very well. No scaling limits or performance or latency issues have been found. The pilot has been equipped with highly useful features, especially job recovery and multitasking, and is supporting many different storage systems. The pilots, broadcasted by several different and robust delivery systems, are ready to receive and handle the user analysis and production jobs of the ATLAS experiment.

References

[1] ATLAS Technical Proposal, CERN/LHCC/94-43 (1994)
[2] Condor Project Homepage, http://www.cs.wisc.edu/condor/
[3] The PanDA homepage, https://twiki.cern.ch/twiki/bin/view/Atlas/PanDA
[4] T. Maeno, "PanDA: Distributed production and distributed analysis system for ATLAS". These proceedings
[5] https://twiki.cern.ch/twiki/bin/view/Atlas/DistributedDataManagement
[6] https://twiki.cern.ch/twiki/bin/view/Atlas/PanDAJobScheduler
[7] Portable Batch System, http://www.pbsgridworks.com/Default.aspx
[8] http://www.cs.wisc.edu/condor/manual/v6.9/5_4Glidein.html
[9] http://services.atlascomp.org/?redirect=PanDAmon