Journal of Physics: Conference Series **396** (2012) 032114 doi:10.1088/1742-6596/396/3/032114

The WorkQueue project - a task queue for the CMS workload management system

S Ryu[1], **S Wakefield**[2]

[1] Fermi National Laboratory, Batavia, Illinois, USA
[2] Imperial College London, London, UK

E-mail: stuart.wakefield@imperial.ac.uk

Abstract. We present the development and first experience of a new component (termed WorkQueue) in the CMS workload management system. This component provides a link between a global request system (Request Manager) and agents (WMAgents) which process requests at compute and storage resources (known as sites). These requests typically consist of creation or processing of a data sample (possibly terabytes in size).

Unlike the standard concept of a task queue, the WorkQueue does not contain fully resolved work units (known typically as jobs in HEP). This would require the WorkQueue to run computationally heavy algorithms that are better suited to run in the WMAgents. Instead the request specifies an algorithm that the WorkQueue uses to split the request into reasonable size chunks (known as elements). An advantage of performing lazy evaluation of an element is that expanding datasets can be accommodated by having job details resolved as late as possible.

The WorkQueue architecture consists of a global WorkQueue which obtains requests from the request system, expands them and forms an element ordering based on the request priority. Each WMAgent contains a local WorkQueue which buffers work close to the agent, this overcomes temporary unavailability of the global WorkQueue and reduces latency for an agent to begin processing. Elements are pulled from the global WorkQueue to the local WorkQueue and into the WMAgent based on the estimate of the amount of work within the element and the resources available to the agent.

WorkQueue is based on CouchDB, a document oriented NoSQL database. The WorkQueue uses the features of CouchDB (map/reduce views and bi-directional replication between distributed instances) to provide a scalable distributed system for managing large queues of work.

The project described here represents an improvement over the old approach to workload management in CMS which involved individual operators feeding requests into agents. This new approach allows for a system where individual WMAgents are transient and can be added or removed from the system as needed.

1. Introduction

The CMS workload management system [1] is responsible for data production and processing for CMS. A large number of computing and storage resources are used in these activities. This model and its possible evolution is described in [2]. These activities are made up of many individual workflows, such as producing simulation data, analysing data from the detector etc. A workflow defines the characteristics of the processing; including the data to be processed the algorithms used and the data to be written out. These workflows vary in the amount of work they require up to, and exceeding, thousands of hours of compute time and terabytes of data.

These resources are provided by 1 Tier-0, 7 Tier-1 and ∼50 Tier-2 sites. $\mathcal{O}(100,000)$ jobs are required to run continuously on these resources to perform CMS's various computing activities. These jobs run the CMS software framework (CMSSW) which is capable of reading, writing and processing CMS data. This software is driven by a configuration file that describes the inputs, outputs and processing to be carried out. It is the job of the workload management system to split the workflow into jobs, configure each job and to submit and track them. The system needs to manage and automate this e.g. identifying new data to run over, cope with job failures etc.

CMS has avoided a central component managing all its distributed computing workflows, believing that independent instances (or agents) allowed for greater scalability and reliability. Agents were given a workflow which they split into jobs that were then submitted and tracked. This approach had the disadvantage that each instance had to be fed its workflows by an operator physically logging into the machine hosting the server and manually running a command. As well as being inefficient this gave an opportunity for human error to creep in and meant there was no reliable record of the workflow that had been run. This also meant that work could only be given to an agent while the operator was available. The lack of communication between agents resulted in an inefficient distribution of work; some agents were idle while others were busy.

The problem with workflow tracking and provenance was solved by the addition of a new component in the workload management system termed the Request Manager (ReqMgr) [1]. When a workflow needs to be run it is uploaded here first, this request as it is now known can then be fed to agents to be run. This provides a central record of what was run and provides a single point where the status of a workflow can be found.

To improve reliability and automation another new system was introduced which takes requests from ReqMgr, assesses priorities and feeds them to the agent (known as WMAgents in CMS [1]) best placed to carry out the work. This system is called the WorkQueue. The WorkQueue functions as a central task queue which allows the agents to be considered transient, allowing instances to be removed or added as needed with no break in activity. This system is illustrated in Figure 1.

2. Features

The WorkQueue contacts ReqMgr to obtain new workflows, which are then expanded into chunks of work and stored in its database. The WorkQueue does not fully resolve the workflow into the individual jobs that the WMAgent will. Performing this algorithm would place a high load on the central server and force it to store and update a large number of items (millions). Instead the expansion is kept deliberately coarse with each element designed to represent a suitably sized batch of work. This ensures the WorkQueue scales and reduces the amount of information flow between the WorkQueue and agents.

WorkQueue's can be joined together to form multi-queue hierarchies, at each WorkQueue the workflow is split into elements that more closely resemble the final state in the agents. So far it has not been necessary to complicate the deployment setup with multiple layers of queue. CMS has deployed one global WorkQueue based at CERN which is used to feed all WMAgents. Each WMAgent contains its own WorkQueue, the local WorkQueue, which is used to obtain work from global and buffer work in the agent before the work is injected into the WMAgent proper. This allows the WMAgent to obtain work quickly when it has free resources and allows the agent to mitigate periods when the global WorkQueue is unavailable.

A workflow controls how the WorkQueue expands it into elements by setting a splitting policy and parameters, see Figure 2. This allows the WorkQueue to adapt to widely varying workflows. Currently the range of policies is quite limited. Workflows requiring input data are split based on groupings (blocks) of input files. In CMS's data management system a block of files are moved around together which allows a reduction in the number of queries needed of the file location

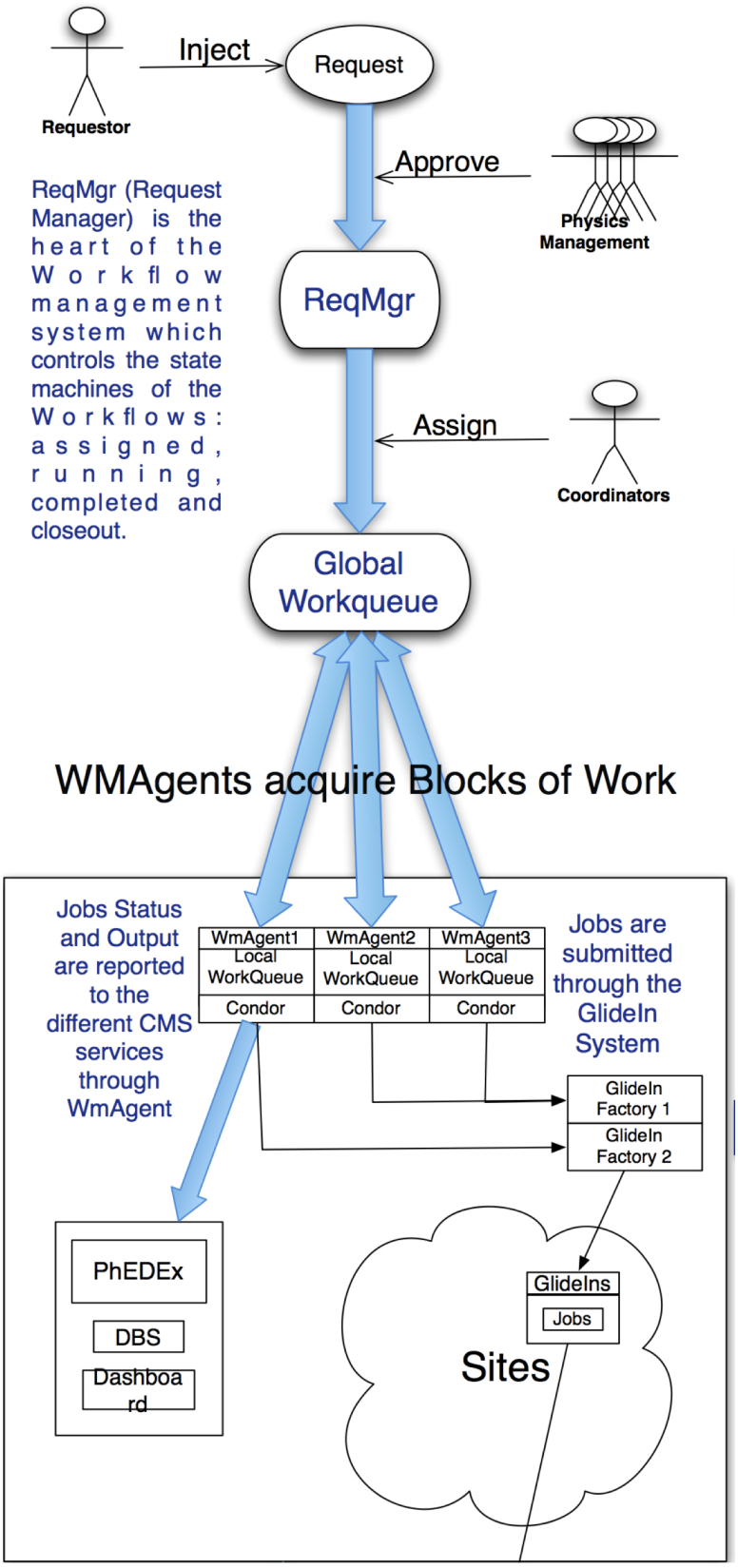

Figure 1. Workload management architecture.

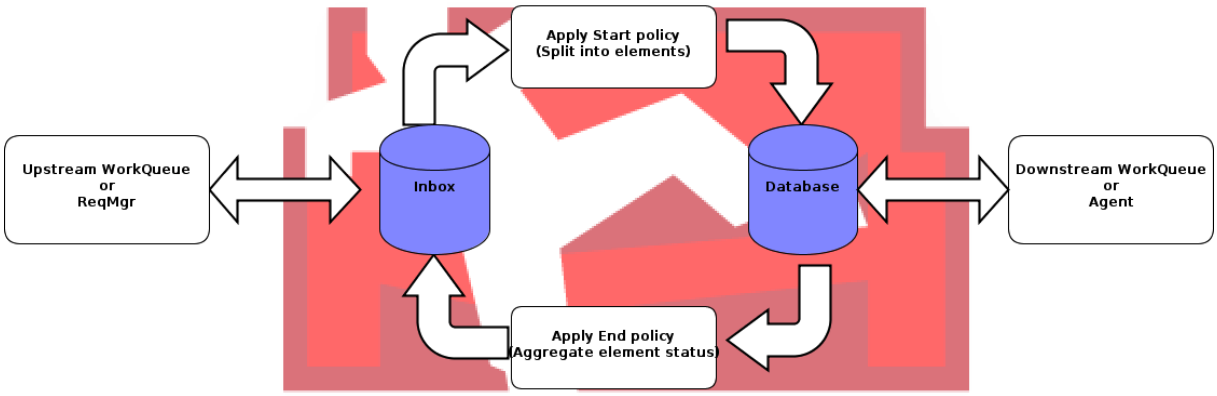

Figure 2. WorkQueue start and end policies.

service. More advanced policies are possible, such as only allowing a subset of a workflow to be run initially to act as a validation step which can catch problem workflows before significant resources have been wasted.

A workflow contains a priority which the WorkQueue uses to schedule elements against each other. To prevent low priority work starving in the queue while high priority obtains resources, the WorkQueue increases an elements relative priority based on the length of time the element has been queued.

Another policy, the end policy, is used to aggregate the status of the elements in order to report the workflow status back. More advanced features have yet to be implemented but could include: the release/cancellation of the workflow based on the validation sample or using failure statistics for the whole workflow as a trigger for stopping processing and reducing any waste of compute resources.

3. Architecture and Technology

The WorkQueue is composed of two internal databases. The Inbox is used to hold incoming workflows before splitting. This is also used to hold the aggregate state of the workflow which can, in the case of the global WorkQueue, then be reported back to the Request Manager. After splitting the resultant WorkQueue elements are saved to the element database where they are available for the local WorkQueue's to work on. In the local WorkQueue the Inbox holds the WorkQueue element acquired from the global queue. As in the global, the splitting is then performed allowing the local queue to either work with the same of smaller chunks of work than global.

The WorkQueue is based on the Apache CouchDB [3] NoSQL document store. This provides both database and web server functionality. The Inbox and main database are implemented as separate CouchDB databases (co-hosted on the same server). The widely varying workflows the WorkQueue has to work with fit well with CouchDB's schema-less document oriented feature set. One of CouchDB's key features is the ability to pre-define queries on the stored data which can use high level languages such as JavaScript, these are known as views. The results from a view are stored and only recomputed when an underlying element changes, this allows them to be performant. The WorkQueue uses over 20 views, which are used for a variety of tasks from essential WorkQueue tasks to provide monitoring web pages.

CouchDB allows for a view to be fed to a server side process along with a list of parameters, this allows dynamic queries to be supported. One of these list functions, as they are known, is used when the local queue in an agent is acquiring work from the global WorkQueue. A view is

generated which lists the available elements in priority order, the list function is then called by the WorkQueue on that view and combined with the site resources available. The list function checks that an element is compatible with the resources provided, e.g. that the site contains all necessary input data and is not explicitly excluded in the workflow and returns the element to the local WorkQueue. The same process is used when the local WorkQueue in the agent retrieves work, from itself, to feed the agent.

CouchDB includes a replication feature which is used to keep the global and local WorkQueues' synchronized. When an element is acquired by a local queue it is marked as belonging to that queue. The CouchDB in the local queue is then configured to continually keep elements synchronized between the two instances.

After an element is injected into the WMAgent the WorkQueue monitors the progress of the resultant jobs. This is saved to the local Inbox and thus replicated back to the global WorkQueue. The global WorkQueue aggregates this over all the elements for a workflow and updates ReqMgr with the resultant progress. While it is doing this the global WorkQueue also monitors the ReqMgr for any attempt to cancel the workflow. If it detects this it marks its elements with a cancellation request, which is then replicated down to the local WorkQueue. When the local WorkQueue detects this request it asks the WMAgent to stop sending out and new jobs and to cancel any already running.

When a workflow has finished processing in the agent, one of its components calls a WorkQueue function that marks the elements in a workflow as complete. The WorkQueue then runs the relevant end policy and updates the relevant Inbox elements. The elements in the main database are then deleted. Once the updated Inbox elements are propagated up to the global WorkQueue, the global WorkQueue runs the relevant end policy and updates its inbox element. The elements in the global WorkQueue are then deleted, this deletion is replicated to the local WorkQueue which removes the elements from the Inbox. The ReqMgr is updated with the final status and the inbox element is deleted.

4. Conclusion

The WorkQueue has been in use for approximately a year. It currently handles almost all organised production and processing for CMS. In this time it has received 3 million writes, ($\sim$ 1 per second). Its adoption has allowed the CMS operations team to increase their efficiency [4]. Computing resource usage also increased due to the streamlined procedure for assigning workflows to agents.

References

[1] Wakefield S *et al. The CMS workload management system* presented at the Conference on Computing in High Energy and Nuclear Physics CHEP2012, New York, USA 2012
[2] Grandi C, Bockelman B, Bonacorsi D, Donvito G, Dykstra D, Fisk I, Hernandez J, Metson S, Sfiligoli I and Wakefield S *Evolution of the Distributed Computing Model of the CMS experiment at the LHC* presented at the Conference on Computing in High Energy and Nuclear Physics CHEP2012, New York, USA 2012
[3] http://couchdb.apache.org/
[4] Fajardo E *et al. A new era for central processing and production in CMS* presented at the Conference on Computing in High Energy and Nuclear Physics CHEP2012, New York, USA 2012