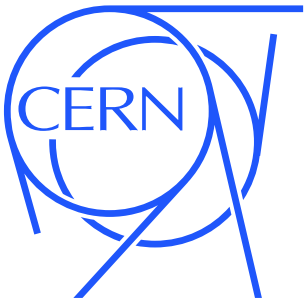

Portable Analysis Environment using Virtualization Technology (WP9)

Co-Pilot: The Distributed Job Execution Framework

Predrag Buncic Artem Harutyunyan

Refers to Co-Pilot 0.1

Technical Report
Version: 0.1 (Draft) March 2011

Abstract

The CernVM Co-Pilot is a framework which allows to integrate cloud and volunteer computing resources into the existing Grid infrastructures and exploit those resources for the execution of Grid jobs. It can also be used to instantiate an independent ad-hoc distributed computing infrastructure on top of virtualized resources.

Contents

1 Overview

The job execution in Grid infrastructures of LHC experiments is done using so-called pilot job approach: experiments come into posession of Grid resrouces using special kind of jobs, known as 'pilot jobs' (or 'job agents', 'pilot agents', 'pilots'), after which they use their own job submission and scheduling systems (e.g. [16, 11, 10, 13]) to deliver and execute Grid users' jobs on those resources.

After landing on the worker nodes of the computing sites the pilot jobs inspect the machine configuration (e.g. check which software packages are available), make necessary preparations for the execution of the real job (e.g. install experiment application packages), report to the experiment scheduling system the configuration of the machine (e.g. available disk space and memory), fetch the real job for execution, fork a process and start execution of the real job within that process.

CernVM Co-Pilot is a framework for execution of LHC experiments' pilot jobs on a wide choice of extra (and so far untapped by LHC experiments) computing resources such as enterprise cloud computing infrastructures (e.g. Amazon EC2 [8]), scientific computing clouds (e.g. Nimbus [15, 19]), and, last but not least, cloud of BOINC ([12]) volunteer PC's. Co-Pilot serves as a gateway between differing pilot job implementations of the LHC experiments and cloud resources. On each experiment's side of the gateway so-called Co-Pilot Adapter service is required. Currently such adapers have been developed for both ALICE and ATLAS, adapters for CMS and LHCb should be produced in order to complete the system.

In order to provide additional computing resources to the LHC experiments, it is important not to expect the physicists who run job production to make changes to their existing scripts or procedures, that is why the Co-Pilot is designed in a way that the integration of the external resources is completely transparent for end-users.

All the code needed to support the Co-Pilot services and to communicate with the Grid services of experiments is included in the CernVM [9, 1] images that we use. The images also contain the ready-to-use LHC experiments' application frameworks.

The Co-Pilot framework (Figure 1.1) consists of the Co-Pilot Agent a lightweight program which runs inside virtual machines deployed on the cloud, and of Co-Pilot Services – components which orchestrate the work of Co-Pilot Agents. Agents and Services communicate by means of Jabber/XMPP [17, 18] instant messaging network using a specially developed protocol. The Co-Pilot Service components are stateless which in conjunction with the use of Jabber/XMPP allows to scale the system in case of high load by just deploying new Service instances and adding them to the existing messaging network .

Co-Pilot framework has two goals. The first goal is to integrate cloud resources with the existing grid infrastructures by seamlessly extending experiment grid pilot job frameworks. The second goal is to provide a toolkit which can be used to quickly

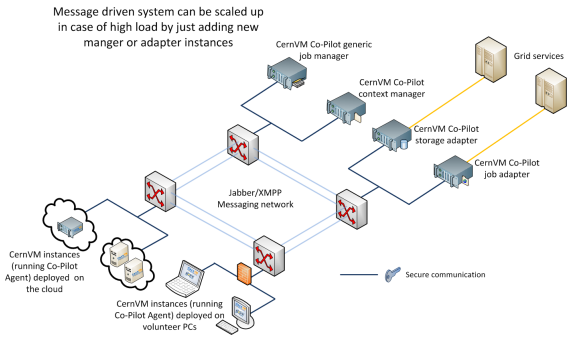

Figure 1.1: The architecture of CernVM Co-Pilot framework

instantiate an ad-hoc distributed computing infrastructure on top of cloud resources. Such a need often arises within the relatively small experiments or research groups (e.g. NA61 experiment [**?**], CERN Theoretical Physics group) which do not have enough resources to deploy and maintain a grid infrastructure.

The components of the Co-Pilot framework are listed in Table 1.1.

Co-Pilot component name	Description
`Agent`	Communicates with service insances and requests a job to execute. Upon receiving the job downloads the input files, executes the job, uploads job output files and reports that the job execution is finished.
`Generic Job and Storage Manager`	Distributes jobs from the internal queue to Agents for execution, provides space for storing the job output
`AliEn Job Manager`	Retrieves jobs from the central task queue of AliEn Grid [16] and sends them to Agents for execution
`AliEn Storage Manager`	Uploads the output of AliEn jobs executed by Agents and finalizes the job status in AliEn Grid.
`PanDA Storage Manager`	Uploads the output of PanDA [10] jobs executed by Agents and finalizes the job status in PanDA Grid

Table 1.1: List of Co-Pilot componentns

2 CernVM Co-Pilot Internals

2.1 Introduction

Agents communicate with services over Jabber/XMPP [17, 18] instant messaging protocol. The messages they exchange are formatted using XML. The messages are enclosed in *<message>* tag, which has three attributes: *'from'* and *'to'* (sender's and receiver's Jabber IDs), as well as the *'id'* tag (unique ID of the message). The content of the message is enclosed in *<body>* tag. All the values inside *<body>* are base64 [14] encoded.

Each message body contains the *'info'* tag. *'info'* has the *'command'* attribute, which contains the command needed to be performed by the peer (e.g. Agent sends 'jobDone' command when it has finished executing the job and uploading files). The *'info'* tag also contains other attributes necessary to perform the command. Below we present the list of commands which agents send along with the appropriate response commands which they expect in response.

Every message by default should be acknowledged by the recipient by sending the *ack* message, with empty *'body'* tag. The ID of the message being acknowledged is set as the value of *'ack'* attribute. Example:

```
<message to='jm@jabber.org'
                    from='agent@jabber.org'
                    ack='1a58119b-5c0e-47e8-8320-b865cc4a502e'
                    >
                   <body> </body>
</message>
```

Listing 2.1: Example *'ack'* message

Acknowledgements can be disabled for arbitrary messages by including a *'noack'* property in the *'message'* tag. This property will signal the recipient not to send an acknowledgement. Example:

```
<message to='mon@jabber.org'
         from='jm@jabber.org/alpha'
         noack='1'>
         <body>
            <info command='__BASE64:cmVwb3J0RXZlbnQ='
                  component='__BASE64:am9ib...scGhh'
                  event='__BASE64:c2Vzc2lvbi5zdGFydA=='/>
         </body>
</message>
```

Listing 2.2: Example message

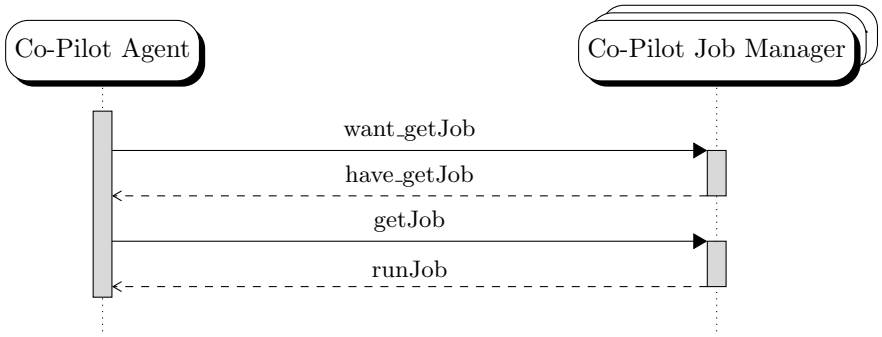

Figure 2.1: Job request commands

2.2 Protocol commands

2.2.1 Job request

When Agent has a slot for running a job it broadcasts to the Job Managers the *'want_getJob'* (sect. 2.2.1.1) request which contains Agent's configuration formatted in JDL [3], hostname and the ip (to be implemented) of the machine where the Agent runs. Upon receiving the message job managers which have jobs for execution send back to the agent *'have_getJob'* 2.2.1.2 command. The agent makes actual job request (*'getJob'* command 2.2.1.3) to the job manager from which the first *'have_getJob'* 2.2.1.2 command has been received. The job manager replies with *'runJob'* 2.2.1.4 command which contains the information about the job. The job request sequence is presented on Figure 2.1. Below the format of the job request commands is described.

2.2.1.1 want_getJob

This message is used to chose a Job Manager to which a *'getJob'* request 2.2.1.3 request eventually is going to be made: the client sends it to the broadcast address and continues the communication with the Job Manager from which responded the quickest. **<info> tag attributes and their values**

- *command* - 'want_getJob'
- *agentHost* - hostname of the agent
- *agentIP* - ip of the agent' to be implemented
- *jdl* - *agent host configuration in JDL format*

Example:

```
<message to='jm@jabber.org' from='agent@jabber.org'>
  <body>
    <info agentHost='_BASE64:Y3ZtYXBwaTI0LmNlcm4uY2g='
```

```
            jdl='_BASE64:CiAgICBbCi . . . . . KICAgIF0='
            command='_BASE64:Z2V0Sm9i'
    />
  </body>
</message>
```

Listing 2.3: Example *'want_getJob'* request

The host configuration in JDL format has the following attributes:

```
[
  Requirements = ( other.Type == "Job" );
  Version = "0";
  Platform = "Linux-i686";
  LocalDiskSpace = 4257288;
  WNHost = "cvmappi24.cern.ch";
  Memory = 1010;
  Type = "machine";
  Uname = "2.6.26.8-0.1.smp.gcc3.4.x86.i686";
  FreeMemory = 683;
  Swap = 127;
  FreeSwap = 127
]
```

Listing 2.4: Example host configuration in JDL format

2.2.1.2 have_getJob

This message is sent by the job manager as a reply to *'want_getJob'* 2.2.1.1 message.

<info> tag attributes and their values

- *command* - 'have_getJob'
- *requireFile* - prerequisite file (optional)

Before making *'getJob'* 2.2.1.3 request Agent has to make sure that a file required by *requireFile* attribute is present on the host. In case when attribute is not present in the message sent by the Job Manager no checks are necessary.

2.2.1.3 getJob

Once the client knows the full JID (Jabber ID + resource) of the Job Manager which has a matching job it sends the real job request which (should eventually result in the *'runJob'* response 2.2.1.4). The *'getJob'* message should has the same format as *'want_getJob'*.

<info> tag attributes and their values

- *command* - 'want_getJob'
- *agentHost* - hostname of the agent
- *agentIP* - ip of the agent' to be implemented
- *jdl - agent host configuration in JDL format*

2.2.1.4 runJob

Upon receiving *'getJob'* 2.2.1.3 request from the client the Job Manager sends to the client the job for execution.

<info> tag attributes and their values

- *command* - 'runJob'
- *jmJobData* - Internal data describing the job (optional)
- *job* - Job information is enclosed in <job> tag, which contains the following attributes:

  chirpUrl - Address of Chirp server from where to get the job input and put the job output (NOTE: Will be changed by the reference to storage adapter which the Agent should ask for the input and upload output)

  inputDir - Directory on Chirp server from which to get the input files of the job

  environment - Environment variables which must be set before job execution

  outputDir - Directory to which to write the output

  packages - List of application software packages needed by the job

  inputFiles - List of job input files

  arguments - Command line arguments of the job execution command

  id - Job ID

  command - Job execution command

  validation - Name of the script for validating the job after it is finished (optional)

If the Job Manager sends the optional *'jmJobData'* property, then the agent will re-send the same value in subsequent requests. If the property isn't provided, Agent will omit it as well.

2.2.2 Job output directory request

When the job execution is finished the client asks the Storage Manager to provide a directory where the output produced by the job must be stored. The client sends a *'want_getJobOutputDir'* 2.2.2.1 request to choose the Job or Storage manager, which is followed by *'getJobOutputDir'* request 2.2.2.3. The client expects *'storeJobOutputDir'* 2.2.2.4 response. Output directory request sequence is given on Figure 2.2.

2.2.2.1 want_getJobOutputDir

This message is used to chose a Job Manager (or a Storage Manager) to which a *'getJobOutputDir'* request 2.2.2.3 request eventually is going to be made: the client

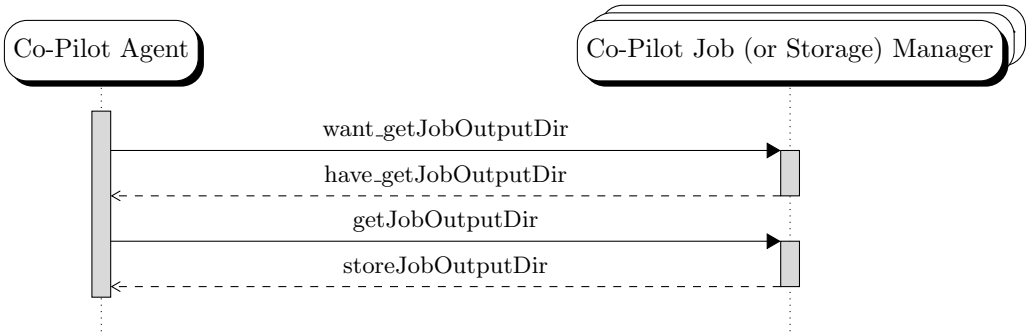

Figure 2.2: Output directory request commands

sends it to the broadcast address and continues the communication with the Job Manager (or a Storage Manager) from which responded the quickest.

<info> tag attributes and their values

- *command* - 'want_getJobOutputDir'
- *agentHost* - Hostname of the client
- *agentIP* - IP of the agent' to be implemented
- *jobID* - Job ID (the one which was received with runJob)
- *jmJobData* - Internal data associated with the job (optional)

If the *'jmJobData'* property was already sent to the Agent (with *'runJob'* command), then the Agent will update its value and accordingly send the new value with subsequent requests.

2.2.2.2 have_getJobOutputDir

This message is sent by the Job or Storage Manager as a reply to *'want_getJobOutputDir'* 2.2.2.1 message.

<info> tag attributes and their values

- *command* - 'have_getJobOutputDir'

2.2.2.3 getJobOutputDir

Clients send the output job directory request to storage server when the job was executed and the output needs to be uploaded.

<info> tag attributes and their values

- *command* - 'getJobOutputDir'
- *agentHost* - Hostname of the client

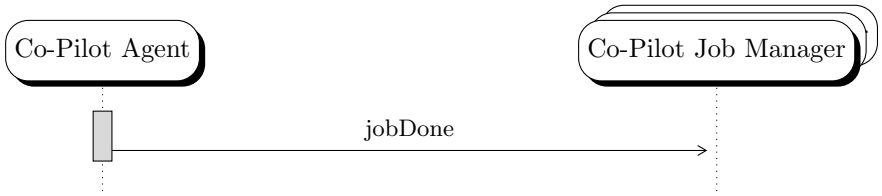

Figure 2.3: Job completion command

- *jobID* - Job ID (the one which was received with runJob)
- *jmJobData* - Job Manager's internal data associated with the job

2.2.2.4 storeJobOutputDir

When Adapter receives an output directory request from the job it creates the directory, sets the write access permission to the Agent, and sends back the directory address.

<info> tag attributes and their values

- *command* - 'storeJobOutputDir'
- *outputChirpUrl* - Chirp server host and port (separated by semicolon)
- *outputDir* - Directory on the chirp server to put files
- *jobID* - Job ID (the one which was received with getJobOutputDir)
- *jmJobData* - Job Manager's internal data associated with the job (optional)

2.2.3 Job completion

2.2.3.1 jobDone

When the job command execution is over and output files are uploaded to the output directory the client reports to the Job Manager that the job is done. Job completion request sequence is presented on Figure 2.3.

<info> tag attributes and their values

- *command* - 'jobDone'
- *exitCode* - Job execution exit code
- *agentHost* - Hostname of the client
- *state* - Job state (e.g. 'done')
- *jobID* - Job ID
- *jmJobData* - Job Manager's internal data associated with the job

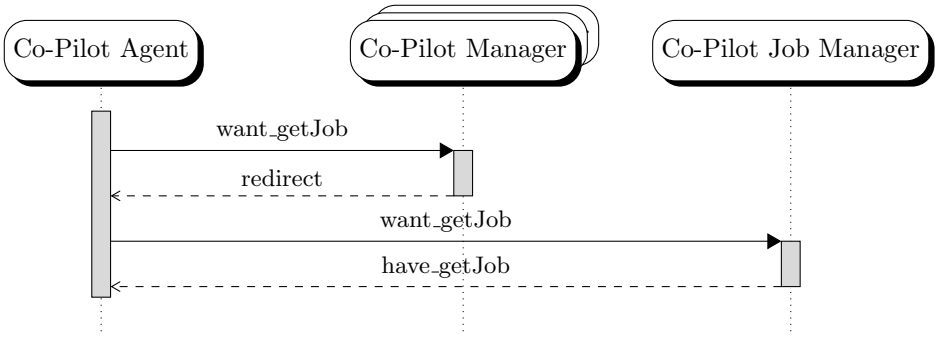

Figure 2.4: Job request commands

2.2.4 Redirection of commands

Messages exchanged by Co-Pilot components can be redirected. This allows having separate instances of Managers for performing different tasks and in the same time configuring clients only with a single communication address. For example one can setup an adapter which is used to retrieve job details and input files and another adapter which is used to upload job execution results and does job finalization (e.g. the use case of ALICE experiment **??**).

2.2.4.1 redirect

<info> tag attributes and their values

- *command* - 'redirect'
- *referral* - Jabber ID of the service to which the redirection should be made
- *info* - Data which should be conveyed to the referral

Example:
The following redirection command received by the Co-Pilot component

```
<message to='agent@cvmappi21.cern.ch' from='jmreal@cvmappi21.cern.ch'>
  <body>
    <info referral='_BASE64:c3RvcmFnZXJlYWxxAY3ZtYXBwaTIxLmNlcm4uY2g='
          command='_BASE64:cmVkaXJlY3Q='>
      <info to='_BASE64:am1yZWFsQGN2bWFwcGkyMS5jZXJuLmNo'
            exitCode='_BASE64:MA=='
            jobID='_BASE64:MzA4MDE3NDE='
            state='_BASE64:ZG9uZQ=='
            jmJobData='_BASE64:c3RhcnRlZEF0PQo='
            command='_BASE64:am9iRG9uZQ==' />
    </info>
  </body>
</message>
```

Listing 2.5: Example *'want_getJob'* request

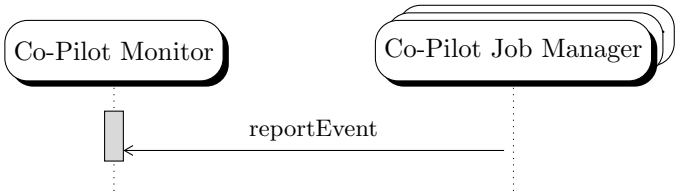

Figure 2.5: Event reporting command

will result into sending the following message to 'storagereal@cvmappi21.cern.ch' (which corresponds to the Base64 decoded value of 'referral' attribute from previous command):

```
<message to='storagereal@cvmappi21.cern.ch' from='agent@cvmappi21.cern.ch'>
  <body>
    <info to='_BASE64:am1yZWFsQGN2bWFwcGkyMS5jZXJuLmNo'
          exitCode='_BASE64:MA=='
          jobID='_BASE64:MzA4MDE3NDE='
          jmJobData='_BASE64:c3RhcnRlZEF0PQo='
          state='_BASE64:ZG9uZQ=='
          command='_BASE64:am9iRG9uZQ==' />
  </body>
</message>
```

Listing 2.6: Example *'want_getJob'* request

2.2.5 Co-Pilot monitoring

Contribution by GSoC 2011 Student J. Lisec

2.2.5.1 reportEvent

To log the events happening in a component, one can send a *'reportEvent'* command as a short blurb about the event.

<info> tag attributes and their values

- *command* - 'reportEvent'
- *component* - identifier of the component, can be unique (e.g. 'jobmanager.generic.jm1') or represent a group of components (e.g.'agent').
- *event* - name of the even that is being reported

2.2.5.2 reportEventDuration

To measure duration of certain events, the *'reportEventDuration'* command can be sent to the Monitor.

<info> tag attributes and their values

- *command* - 'reportEventDuration'
- *component* - identifier of the component. It can be either unique (e.g. 'jobmanager.generic.jm1') or represent a group of components (e.g. 'agent')
- *event* - name of the event that is being reported
- *duration* - duration of the event in milliseconds

2.2.5.3 reportEventValue

Unlike *'reportEvent'* whose values are sampled and sent every few seconds to the database, this command reports the given value immediately. Example use cases include monitoring system resources e.g. CPU load or network bandwidth.

<info> tag attributes and their values

- *command* - 'reportEventValue'
- *component* - identifier of the component. It can be either unique (e.g. 'jobmanager.generic.jm1') or represent a group of components (e.g. 'agent')
- *event* - name of the event that is being reported
- *value* - numerical value representing the event

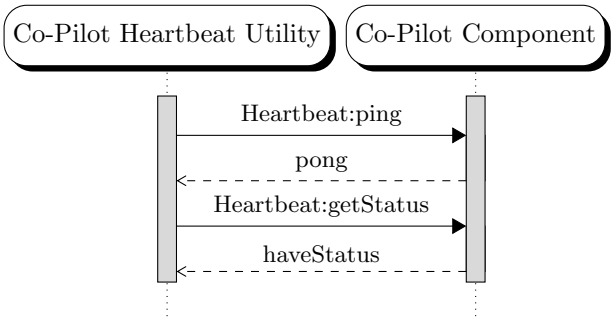

Figure 2.6: System introspection commands

2.2.6 System introspection

Co-Pilot protocol defines a few simple commands which allow remote system introspection. For example, it is possible to query the system load or disk usage of a system on which the component is running.

2.2.6.1 Heartbeat:getStatus

To request the information of a particular system component one sends the *'getStatus'* command. This command is usually sent by the `copilot-heartbeat` (see 4.5.4) command.

<info> tag attributes and their values

- *command* - 'Heartbeat:getStatus'
- *component* - name of the component whose status is needed (eg. 'systemLoad', 'diskUsage', etc.)

Accepted values of `component` attribute

- *systemLoad*
- *diskUsage*
- *memoryUsage*
- *runningProcesses*

The flow of message exchange is described on Figure 2.6.

2.2.6.2 haveStatus

This message is sent as a reply to *'Heartbeat:getStatus'*.

<info> tag attributes and their values

- *command* - 'haveStatus'

- *component* - component whose status is being reported
- *status* - a string describing the component's status

2.2.6.3 Heartbeat:ping

Heartbeat service may occasional send the ping command which just tests the availability of the receiver.

<info> tag attributes and their values

- *command* - 'Heartbeat:ping'

2.2.6.4 pong

This command is used as a reply to the *'Heartbeat:ping'* command.

<info> tag attributes and their values

- *command* - 'pong'

2.2.7 Secure mode of operation

Co-Pilot components can also operate in secure mode. In that case all messages they exchange are encrypted using AES symmetric encryption algorithm. Encryption is done using 256 bit key, which is generated by the client and is sent along the request to the Manager. The session key itself is encrypted using RSA algorithm with Adapter's public key (so it can be decrypted only using corresponding private key. Adapter encrypts the replies which it sends to agent using the key it got during the request.

<info> tag attributes and their values

- *session_key* - The session key which was used to encrypt the message
- *info* - The actual message

2.3 Co-Pilot Monitoring

2.3.1 Concepts

Co-Pilot's monitoring subsystem is using Graphite[5] for data storage and graphing.

Graphite's underlying database, Whisper, also called round-robin database, stores its data into files of predetermined size. The size of a database is calculated by data retention for each metric, for instance, it can store two weeks worth data at the one minute precision, meaning that there will be a data point stored for every minute in those two weeks.

Metric (event in Co-Pilot jargon), is a name of a data being collected and it's translated into a path by Whisper. A path is constituted of individual path components which are delimited with a full stop.

```
copilot.jobmanager.generic.default.system.net.in.eth0
copilot.jobmanager.generic.default.system.net.out.eth0
```

Listing 2.7: Example Graphite paths

In Graphite's web interface and on the file system full stops are used to represent directories. Replacing a path component with an asterisk will combine several metrics into a single graph.

```
copilot.jobmanager.generic.default.system.net.*.eth0
```

Listing 2.8: Combined Graphite path

For durations (events reported with *'reportEventDuration'* 2.2.5.2), Co-Pilot Monitor will store the following data for each sample along with the duration:

- *(event).start* - logged when the event starts
- *(event).end* - logged when measuring stops
- *(event).avg* - arithmetical mean of the duration
- *(event).gmean* - geometrical mean of the duration
- *(event).min* - shortest durations
- *(event).max* - longest durations

2.3.2 Monitored events

The following list contains paths of events which are being reported by Co-Pilot Monitor:

- *copilot.monitor.updates* - number of updates logged per sample
- *copilot.jobmanager.(type).(jid).job.succeeded* - job completed successfully
- *copilot.jobmanager.(type).(jid).job.failed* - job exited with an error
- *copilot.jobmanager.(type).(jid).queuedJobs* - size of the job queue
- *copilot.jobmanager.(type).(jid).error.invalidJobId* - Manager received an request with non-existing job
- *copilot.jobmanager.(type).(jid).error.emptyQueue* - the job queue was empty when either *'want_getJob'* *'getJob'* command was received
- *copilot.jobmanager.(type).(jid).system.disk.available.(disk)* - available disk space (in megabytes)
- *copilot.jobmanager.(type).(jid).system.disk.used.(disk)* - used disk space (in megabytes)

- *copilot.jobmanager.(type).(jid).system.load.1min* - system load in past minute
- *copilot.jobmanager.(type).(jid).system.load.5min* - average load in past five minutes
- *copilot.jobmanager.(type).(jid).system.load.15min* - average load in past 15 minutes
- *copilot.jobmanager.(type).(jid).system.net.in.(interface)* - incoming network bandwidth (in megabytes)
- *copilot.jobmanager.(type).(jid).system.net.out.(interface)* - outgoing network bandwidth (in megabytes)
- *copilot.jobmanager.(type).(jid).system.ram.used.(mem—swap)* - used memory (in megabytes)
- *copilot.jobmanager.(type).(jid).system.ram.available.(mem—swap)* - available memory (in megabytes)

Note that *(type)* represents a type of a Job Manager that has reported the event (Generic, AliEn or PanDA) and that *(jid)* is the XMPP resource of a particular Job Manager instance.

3 Use cases

Co-Pilot Managers can either be used to act as a proxy between Co-Pilot Agent and the grid infrastructures, or can be used on their own. Grid infrastructures of experiments differ in their architecture, the Co-Pilot components, however, are flexible enough to accommodate those differences.

3.1 Interfacing Co-Pilot with the ALICE grid

The execution of the ALICE experiment grid jobs is done in the following way. The Co-Pilot Agent communicates with the AliEn Job Manager grid service [16], which in turn contacts the central job queue of the AliEn Grid, and requests a job to execute. When it gets a job, it downloads the job input files from the AliEn File Catalogue and makes the files available to download for the Co-Pilot Agent. The Agent downloads the input files and runs the job execution commands. After the commands have been executed, the Agent contacts the Co-Pilot AliEn Storage Manager component and uploads the results to the machine where the Storage Manager runs. The Storage Manager then contacts the AliEn central services, uploads the results of the execution to the AliEn Storage Element specified in the job description, registers the files in the File Catalogue and marks the final status of the job in the AliEn central database.

3.2 Interfacing Co-Pilot with the ATLAS grid

In case of the ATLAS experiment the Co-Pilot agent contacts the Co-Pilot Generic Job and Storage manager, which instructs it to execute a specially configured PanDA pilot [10]. After the execution, the PanDA pilot gets a job from the central queue and executes it. Once the job execution is over the Co-Pilot Agent uploads the results to the Co-Pilot PanDA Storage Manager, which uploads the output to the grid storage, registers the files in the File Catalogue and finalizes the job status.

3.3 Standalone use of Co-Pilot

In case of the CERN Theoretical Physics Group Monte-Carlo simulation application, the Co-Pilot framework is used as a distributed computing platform. The jobs are submitted to the Co-Pilot Generic Job and Storage Manager queue by the operator, and are picked up and executed by the Co-Pilot Agent. After the execution is completed the Agent contacts the Generic Job and Storage Manager again and uploads the results.

4 Deployment

4.1 Prerequisites

4.1.1 XMPP server

Co-Pilot components use XMPP to communicate. We recommend using ejabberd [4]. Ejabberd sources and documentation can be obtained from http://www.ejabberd.im/.

4.1.2 Chirp server

Co-Pilot components exchange files using Chirp[2]. Chirp sources and documentation can be obtained from http://nd.edu/ ccl/software/chirp/. Chirp server should be running on both Job (or Storage) Manager host.

```
$ mkdir -p JM_CHIRP_WORK_DIR
$ chirp_server -i non_root_user -u - --d all -r JM_CHIRP_WORK_DIR
```

Listing 4.1: Starting Chirp server

Where *JM_CHIRP_WORK_DIR* should coincide with the directory defined as *JM_CHIRP_WORK_DIR* in Manager's configuration file (see 4.2.2).

4.1.3 Redis server

Co-Pilot Generic Job Manager uses Redis[7] for storing the job queue. Redis sources and documentation can be obtained from http://redis.io/.

4.1.4 MongoDB server

Co-Pilot Monitor and Co-Pilot Dashboard use MongoDB[6] for storing details about Co-Pilot agents and Dashboard settings. Binary distribution of MongoDB can be obtained from http://mongodb.org/downloads.

4.1.5 Perl modules

Co-Pilot is written in Perl programming language and depends on a list of 3rd party Perl modules. On CernVM, the installation of these modules will be triggered automatically during the installation of Co-Pilot . To install all the dependencies on CernVM manually one can trigger the installation of perl-Component-Copilot package (all Co-Pilot components depend on it):

```
$ sudo conary update perl-Copilot
```

Listing 4.2: Installing Co-Pilot dependencies on CernVM

Running Co-Pilot Generic Job Manager requires manual installation of Perl modules for Redis and MongoDB using CPAN:

```
$ sudo cpan CPAN
['no' to manual configuration]
$ sudo cpan
['yes' to automatic configuration]
cpan[1]> reload cpan
cpan[2]> reload index
cpan[3]> install Redis
cpan[4]> notest install MongoDB
```

Listing 4.3: Installing Perl modules

Installation procedure is interactive and depending on target system it might take some time. When asked if dependencies should be installed, answer with yes. Tests won't be performed for the MongoDB library because they require an up-and-running installation of MongoDB.

4.1.6 Graphite

4.1.6.1 Installing Graphite

Graphite is used for collecting statistics and it powers Co-Pilot Monitor. Graphite sources and documentation can be obtained from http://graphite.wikidot.com/.

Graphite depends on the following Python libraries:

- PySQLite2 (comes with Python 2.5 or newer)
- ctypes (comes with Python 2.5 or newer)
- hashlib (comes with Python 2.5 or newer)
- Zope.Interface (3.7.0)
- Twisted (11.1.0)
- Gevent (0.13.6)
- Gunicorn (0.11.0)
- Django (1.3)
- django-tagging (0.3.1)
- PyMongo (2.2)
- PyCairo (available in CernVM's package manager)

A script is provided in `copilot-monitor/utils/install-graphite.sh` which downloads and installs all Graphite's prerequisites. A list of URLs from which the packages can be manually obtained is provided in the file as well.

There are some known problems with Graphite being installed via Python's package manager, which is why you are advised to it from source:

```
$ wget http://github.com/downloads/graphite-project/whisper/whisper-0.9.10.tar.gz
$ tar xf whisper-0.9.10.tar.gz && cd whisper-0.9.10
$ sudo python setup.py install
$ wget http://github.com/downloads/graphite-project/carbon/carbon-0.9.10.tar.gz
$ tar xf carbon-0.9.10.tar.gz && cd carbon-0.9.10
$ sudo python setup.py install
$ wget http://github.com/downloads/graphite-project/graphite-web/graphite-web-0.9.10.tar.gz
$ tar xf graphite-web-0.9.10.tar.gz && cd graphite-web-0.9.10
$ sudo python setup.py install
```

Listing 4.4: Installing Graphite

4.1.6.2 Configuration

Before using Graphite's web interface, an administrator account has to be created:

```
$ cd /opt/graphite/webapp/graphite
$ sudo python manage.py syncdb
```

Listing 4.5: Creating admin users

Recommended configuration files for Graphite and Whisper are provided in `copilot-monitor/util/config`. Note that you may have to adjust the names of hard disks and network interfaces mentioned in configuration files according to your system. It is also recommended to change the ownership of `/opt/graphite` directory to a non-root user.

4.1.6.3 Troubleshooting

In some cases it might happen that not all of the Graphite's files were copied over to the installation directory. To verify your Graphite installation make sure that both `carbon` and `twisted` directories exist in `/opt/graphite/lib`. Non-functioning aggregation service (`carbon-aggregator.py`) can be one of the indicators of the broken installation as well.

To fix the problem, make sure that user account which will run Graphite processes has read and write access to following directories:

```
$ chown -R manager:manager /opt/graphite
$ chown -R manager:manager /usr/lib64/python2.4/site-packages/Twisted*
```

Listing 4.6: Adjusting directory permissions

And as a final step remove generated Python bytecode files if Twisted is still reporting an error.

```
$ rm /opt/graphite/lib/carbon/aggregator/*.pyc
```

Listing 4.7: Removing Python bytecode files

4.2 Deployment of Co-Pilot Generic Job and Storage Manager

4.2.1 Installing the Job Manager

To obtain Job Manager sources and install it do

```
$ svn co https://cernvm.cern.ch/project/svn/copilot/trunk copilot
$ svn co \
https://cernvm.cern.ch/project/svn/copilot-jobmanager-generic/trunk \
copilot-jobmanager-generic
$ cd copilot
$ perl Makefile.PL
$ make
$ sudo make install
$ cd ../copilot-jobmanager-generic
$ perl Makefile.PL
$ make
$ sudo make install-all
```

Listing 4.8: Co-Pilot Generic Job Manager installation

4.2.2 Configuring the Job Manager

The default location of Job Manager configuration is in `/etc/copilot/copilot-jobmanager-generic.conf`. Config file has a simple format (VARIABLE_NAME WHITESPACE VARIABLE_VALUE).
The value of the following variables must be set before the Job Manager can be started:

- *JM_JABBER_SERVER* - address of your Jabber/XMPP server (e.g. xmpp.cern.ch)
- *JM_JABBER_DOMAIN* - domain which your Jabber/XMPP server is configured to serve (e.g. xmpp.cern.ch)
- *JM_JABBER_ID* - Jabber ID of the Job Manager (ID should be registered on your jabber server)
- *JM_JABBER_PASSWORD* - password for authenticating as JM_JABBER_ID on JM_JABBER_SERVER
- *JM_JABBER_RESOURCE* - Jabber resource id (e.g. firstcopilotjm, required for Co-Pilot Monitor)
- *JM_LOGGER_CONFIG_FILE* - log file format configuration (e.g. `/etc/copilot/loggerConf/jobmanager`

- *JM_CHIRP_SERVER* - address of the server on which Chirp is running
- *JM_CHIRP_WORK_DIR* - directory which is 'exported' by the Chirp server
- *JM_DONE_JOB_DIR* - directory where the results of the done jobs are saved
- *JM_REDIS_SERVER* - machine on which Redis DB is running (e.g. localhost)
- *JM_WAITING_JOBS_LIST* - Name of the list in the Redis DB containing waiting jobs' IDs (defaults to 'waiting_jobs')
- *JM_MONGO_SERVER* - machine on which MongoDB is running (e.g. localhost; optional)
- *JM_JOB_REQUIRE_FILE* - Name of the file which is required to be present on the Agent machine which receives a job from this Job Manager (optional). For details see 2.2.1.2.
- *JM_QUEUE_ONLY_MODE_ON* - Set to '1' if you want the Job Manager to only act as a job queue (i.e. ignore messages related to storing job results)
- *JM_STORAGE_ONLY_MODE_ON* - Set to '1' if you want the Job Manager to only act as a storage service (i.e. ignore messages related to requesting new jobs)
- *MON_JABBER_ADDRESS* - Jabber address of the Monitor component (e.g mon@xmpp.cern.ch; optional)
- *HB_CHAT_SERVER* - domain name of the XMPP MUC server (e.g. conference.xmpp.cern.ch; optional)
- *HB_CHAT_ROOM* - chat room used for Heartbeat service (e.g. pilots-lounge; optional)

You might also want to take a look at the log format definition file (specified by JM_LOGGER_CONFIG_FILE).

4.2.3 Starting the Job Manager

Job manager can be started with `copilot-jobmanager-generic`. The output of the command should look like this:

```
$ copilot-jobmanager-generic
[Fri Mar 25 16:18:57 2011] [info] The component does not need to be waken up.
[Fri Mar 25 16:18:57 2011] [info] The security module does not need to be waken up.
```

Listing 4.9: Starting Co-Pilot Generic Job Manager

4.3 Deployment of Co-Pilot Agent

4.3.1 Installing the Agent

To obtain Agent sources and install it do:

```
$ svn co https://cernvm.cern.ch/project/svn/copilot/trunk copilot
$ svn co \
https://cernvm.cern.ch/project/svn/copilot-agent/trunk \
copilot-agent
$ cd copilot-agent
$ perl Makefile.PL
$ make
$ sudo make install-all
```

Listing 4.10: Co-Pilot Agent installation

4.3.2 Configuring the Agent

Agent configuration file is located in `/etc/copilot/copilot-agent.conf`. Configuration file has a simple format (VARIABLE_NAME WHITESPACE VARIABLE_VALUE). The value of the following variables must be set before the Agent can be started:

- *JM_JABBER_ADDRESS* - Jabber ID of the Job Manager (jobmanager@xmpp.cern.ch)
- *AGENT_JABBER_SERVER* - address of your Jabber/XMPP server (e.g. xmpp.cern.ch)
- *AGENT_JABBER_DOMAIN* - domain which your Jabber/XMPP server is configured to serve (e.g. xmpp.cern.ch)
- *AGENT_JABBER_ID* - Jabber ID of the Agent (ID should be registered on your jabber server)
- *AGENT_JABBER_PASSWORD* - password for authenticating as `AGENT_JABBER_ID` on `AGENT_JABBER_SERVER`
- *AGENT_LOGGER_CONFIG_FILE* - log file format configuration (e.g. `/etc/copilot/loggerConf/agent`
- *AGENT_WORK_DIR* - working directory of the Agent (should be writeable)
- *HB_CHAT_SERVER* - domain name of the XMPP MUC server (e.g. conference.xmpp.cern.ch; optional)
- *HB_CHAT_ROOM* - chat room used for Heartbeat service (e.g. pilots-lounge; optional)

You might also want to take a look at the log format definition file (specified by JM_LOGGER_CONFIG_FILE).

4.3.3 Starting the Agent

Agent can be started with copilot-agent. The output of the command should look like this:

```
$ copilot-agent
[Fri Mar 25 17:09:02 2011] [info] Waking the component up.
[Fri Mar 25 17:09:03 2011] [info] The security module does not need to be waken up.
[Fri Mar 25 17:09:03 2011] [info] Component was waken up. Asking job manager for a job
[Fri Mar 25 17:09:03 2011] [info] Asking jobmanager@xmpp.cern.ch for an address of the job mana
[Fri Mar 25 17:09:03 2011] [debug] Sending message to jobmanager@xmpp.cern.ch the component (Ms
[Fri Mar 25 17:09:03 2011] [debug] Got ACK for 47b06aea-2222-4f41-be1b-507de80de51c
```

Listing 4.11: Starting Co-Pilot Agent

4.4 Deployment of Co-Pilot Monitor

4.4.1 Installing Co-Pilot Monitor

To obtain Monitor's source code and install it:

```
$ svn co https://cernvm-copilot-monitor.googlecode.com/svn/trunk/copilot-monitor\
copilot-monitor
$ cd copilot-monitor
$ perl Makefile.PL
$ make
$ sudo make install-all
```

Listing 4.12: Installing the Monitor

4.4.2 Configuring Co-Pilot Monitor

Monitor's configuration file is located in `/etc/copilot/copilot-monitor.conf` and follows the same format as other Co-Pilot components: (VARIABLE_NAME WHITESPACE VARIABLE_VALUE).

- *MON_JABBER_SERVER* - address of your Jabber/XMPP server (e.g. xmpp.cern.ch)
- *MON_JABBER_DOMAIN* - domain which your Jabber/XMPP server is configured to serve (e.g. xmpp.cern.ch)
- *MON_JABBER_ID* - Jabber ID of the Job Manager (ID should be registered on your jabber server)
- *MON_JABBER_PASSWORD* - password for authenticating as `MON_JABBER_ID` on `MON_JABBER_SERVER`
- *CARBON_SERVER* - address of Carbon database server
- *CARBON_PORT* - port on which Carbon aggregation service is running (e.g. 2023)

4.4.3 Starting the Monitor

Monitor can be started with the `copilot-monitor` command. The output should look like this:

```
$ copilot-monitor
[Thu Aug 11 19:52:58 2011] [info] The component does not need to be waken up.
[Thu Aug 11 19:52:58 2011] [info] The security module does not need to be waken up.
[Thu Aug 11 19:53:06 2011] [info] Sent 0 updates to Carbon.
```

Listing 4.13: Starting Co-Pilot Monitor

4.4.4 Ejabberd Module

4.4.4.1 Installing ejabberd Module

As a part of the monitoring solution, an ejabberd module is available for collecting the data like number of connected users or their geographical location.

```
$ cd copilot/src/copilot-ejabberd-module
$ ./configure
$ make get-deps
$ sudo install-deps
$ make compile
$ sudo make install
```

Listing 4.14: Installing ejabberd module

4.4.4.2 Configuring ejabberd Module

In order to get ejabberd to load the module, ejabberd's configuration file needs to be updated. At the end of list of modules in `/etc/ejabberd/ejabberd.cfg` you need to add:

```
{modules, [
  ...
  {mod_copilot, [
    {mongodb, {"localhost", 27017}},
    {monitor_jid, "monitor@localhost"},
  ]}
]}.
```

Listing 4.15: Updated ejabberd.cfg

After modifying the configuration file, the module could be either restarted or the module can be dynamically loaded through ejabberds console:

```
$ /sbin/ejabberdctl debug
1> ejabberd_config:load_file("/etc/ejabberd/ejabberd.cfg").
2> l(mod_copilot).
{module, mod_copilot}
3> [press Control + C twice to quit]
```

Listing 4.16: Loading mod_copilot

4.4.5 Installing Co-Pilot Dashboard

Monitor comes with a simplified dashboard interface for instant overview of the system. The dashboard is meant to augment the features provided by Graphite's web interface.

To obtain the source code and install it:

```
$ cd copilot/src/copilot-dashboard
$ python setup.py build
$ sudo python setup.py install
```

Listing 4.17: Installing the Dashboard

4.4.6 Configuring Co-Pilot Dashboard

Example configuration file, `copilot-dashboard.conf.example` has been provided in the source tree, the file should be manually copied to `/etc/copilot/copilot-dashboard.conf` and it follows the same format as other Co-Pilot components (VARIABLE_NAME WHITESPACE VARIABLE_VALUE):

- *DASH_SERVER_HOST* - address on which the web server will run (e.g. 0.0.0.0 or localhost)
- *DASH_SERVER_PORT* - port on which the web server will be available (default: 3274)
- *DASH_GRAPHITE_HOST* - address of Graphite's web interface (e.g. localhost)
- *DASH_GRAPHITE_PORT* - port on which Graphite's web interface is running (e.g. 8000)
- *DASH_MONGODB_HOST* - address on which MongoDB is running (eg. localhost)
- *DASH_MONGODB_PORT* - port on which MongoDB is running (default: 27017)
- *DASH_MONGODB_DB* - name of the MongoDB database used by the Dashboard (default: copilot)
- *DASH_GMAPS_API_KEY* - API key for Google Maps (can be obtained at https://code.google.com/apis/console)

4.4.7 Starting Co-Pilot Dashboard

Dashboard can be started with the `copilot-dashboard` command. By default, the web server will be bound to 0.0.0.0 on port 3274.

```
$ copilot-dashboard
Starting Co-Pilot Dashboard on 0.0.0.0:3274...
```

Listing 4.18: Starting Co-Pilot

4.5 Deployment of Co-Pilot Heartbeat

4.5.1 Installing Co-Pilot Heartbeat

To obtain source code of the utility and to install it:

```
$ svn co https://cernvm-copilot-monitor.googlecode.com/svn/trunk/copilot-heartbeat\
copilot-heartbeat
$ cd copilot-heartbeat
$ perl Makefile.PL
$ make
$ sudo make install-all
```

Listing 4.19: Installing the Monitor

4.5.2 Configuring Co-Pilot Heartbeat

Heartbeat's configuration file is located in `/etc/copilot/copilot-heartbeat.conf` and follows the same format as other Co-Pilot components: (VARIABLE_NAME WHITESPACE VARIABLE_VALUE).

- *HB_JABBER_SERVER* - address of your Jabber/XMPP server (e.g. xmpp.cern.ch)
- *HB_JABBER_DOMAIN* - domain which your Jabber/XMPP server is configured to serve (e.g. xmpp.cern.ch)
- *HB_JABBER_ID* - Jabber ID of the Job Manager (ID should be registered on your jabber server)
- *HB_JABBER_PASSWORD* - password for authenticating as MON_JABBER_ID on MON_JABBER_SERVER
- *HB_CHAT_SERVER* - domain name of the XMPP MUC server (e.g. conference.xmpp.cern.ch)
- *HB_CHAT_ROOM* - chat room used for Heartbeat service (e.g. pilots-lounge)

4.5.3 Configuring XMPP server

Co-Pilot Heartbeat service requires a working MUC (Multi User Chat) server. Example configuration of the MUC module for Ejabberd is provided below.

```
{mod_muc, [
          {host, "conference.@HOST@"},
          {access, muc},
          {access_create, muc_create},
          {access_persistent, muc_create},
          {access_admin, muc_admin},
          {history_size, 0},
          {min_presence_interval, 60},
          {logging, false}
```

```
        ]},
```

Listing 4.20: Example configuration of MUC module

4.5.4 Using command-line utility

The command-line utility sends commands to components that are connected to the Heartbeat chat room and thus allows remote introspection of the system.

```
copilot-heartbeat [OPTIONS] [COMMAND] [JIDs]
```

Listing 4.21: Arguments of copilot-heartbeat utility

You can use command line flags to override XMPP settings from the configuration file.

```
$ copilot-heartbeat --chat-room=dev-env list
```

Listing 4.22: Overriding name of the chat room

List of supported commands can be seen with `--help` flag, or refer to section 2.2.6.1.

To send the command to a specific component you can specify its full JID, a room JID or just its room nickname. If no JIDs are specified then the command will be sent to every component connected to the chat room.

```
$ copilot-heartbeat systemLoad \
jmgeneric@xmpp.cern.ch/alpha \
pilots-lounge@conference.xmpp.cern.ch/e48b-42d9 \
25af-ecf4
Room nick    1min    5min    15min
alpha        0.90    0.65    0.52
e48b-42d9    1.51    1.23    1.13
24af-ecf4    1.50    0.95    0.40
```

Listing 4.23: Requesting system load from several components

4.6 Test Job submission

Jobs can be submitted with the *copilot-job-submit* command provided in *copilot-util* package.

```
$ svn co https://cernvm.cern.ch/project/svn/copilot-util/trunk copilot-util
$ perl Makefile.PL
$ make
$ sudo make install-all
```

Listing 4.24: Obtaining copilot-util package from SVN

The `copilot-job-submit` accepts the following arguments:

- `--command` - file which will be executed by the agents

- `--packages` - list of packages needed for the job to run
- `--chirp-server` - address of the Chirp server used by the job manager
- `--chirp-dir` - path to the directory exposed by local Chirp server

Let us set up a simple script which will print a sum of two numbers. Jobs can be executable files, binaries or scripts. In this case we will use a simple shell script:

```
#!/bin/bash

echo "2 + 2" | bc
```

Listing 4.25: Test job (`calculator.sh`)

This script can be submitted to Co-Pilot as a job with `copilot-job-submit` command:

```
$ copilot-job-submit \
--command=calculator.sh \
--chirp-server=cervnvma12.cern.ch \
--chirp-dir=JM_CHIRP_WORK_DIR
```

Listing 4.26: Submitting a job

The value of the `--chirp-server` flag will be provided to agents as the address of the machine where the Chirp server is running. By default `copilot-job-submit` sets it to the hostname of the machine from where it was executed. `JM_CHIRP_WORK_DIR` should coincide with the directory defined as `JM_CHIRP_WORK_DIR` in Manager's configuration file (see 4.2.2). After the job is fetched by the Co-Pilot Agent and executed, its results should be available in `JM_DONE_JOB_DIR` directory (see 4.2.2).

```
$ cd JM_DONE_JOB_DIR/[Job ID]
$ tar xvf [Job ID].tgz
$ cat [Job ID]/stdout
4
```

Listing 4.27: Inspecting job results

Bibliography

[1] Cernvm project. http://cernvm.cern.ch/.

[2] Chirp. http://nd.edu/~ccl/software/chirp/.

[3] Condor classified advertisements. http://www.cs.wisc.edu/condor/classad/.

[4] Erlang jabber/xmpp daemon. http://www.ejabberd.im/.

[5] Graphite. http://graphite.wikidot.com/.

[6] Mongodb. http://mongodb.org.

[7] Redis. http://redis.io.

[8] Amazon. Elastic compute cloud (amazon ec2) service. http://aws.amazon.com/ec2/.

[9] Predrag Buncic, Jakob Blomer, Pere Mato, Carlos Aguado Sanchez, Leandro Franco, and Steffen Klemer. Cernvm - a virtual appliance for lhc applications. In *Proceedings of the XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT08)*, 2008.

[10] ATLAS Collaboration. Panda. http://iopscience.iop.org/1742-6596/119/6/062036.

[11] LHCb Collaboration. DIRAC, A Community Grid Solution. In *Computing in High Energy Physics (CHEP)*, 2007.

[12] Anderson D.P. Boinc: A system for public-resource computing and storage. *Proc. of 5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, 2004.

[13] D. Spiga et al. The cms remote analysis builder (crab). *LNCS*, 4873:580–586, 2007.

[14] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 3548 (Informational), July 2003. Obsoleted by RFC 4648.

[15] Keahey K. and Freeman. T. Science clouds: Early experiences in cloud computing for scientific applications. In *Cloud Computing and Its Applications 2008 (CCA-08)*, Chicago, IL, October 2008.

[16] Saiz P., Aphecetche L., Buncic P. Piskac R., Revsbech J.-E., and Sego V. Alien resource brokers. *Nuclear Instruments and Methods in Physics Research*, 502(2-3), 2003.

[17] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004.

[18] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921 (Proposed Standard), October 2004.

[19] Freeman T. and Keahey K. Science clouds. http://www.scienceclouds.org/.

Index