Journal of Physics: Conference Series **219** (2010) 052025 doi:10.1088/1742-6596/219/5/052025

CluMan – Cluster Management toolsuit

Miroslav Šiket, Marián Babik, Sebastian Lopienski, Filipe David Borba Manana

CERN, 385, Route de Meyrin, CH-1217 Genéve 21, Switzerland

Miroslav.Siket@cern.ch, Sebastian.Lopienski@cern.ch,
Filipe.David.Borba.Manana@cern.ch, Marian.Babik@cern.ch

Abstract. LHC computing requirements are such that the number of CPU and storage nodes, and the complexity of the services to be managed are bringing new challenges. Operations like checking configuration consistency, executing actions on nodes, moving them between clusters etc. are very frequent. These scaling challenges are the basis for CluMan, a new cluster management tool being designed and developed at CERN. High-density displays such as heat maps, grids or color maps are more and more commonly used in various applications like data visualization or monitoring systems. They allow humans to see, interpret and understand complex and detailed information at a glance. We propose to present the ideas behind the CluMan project, and to show how high density displays are used to help service managers to understand, manage and control the state and behavior of their clusters.

1. Introduction

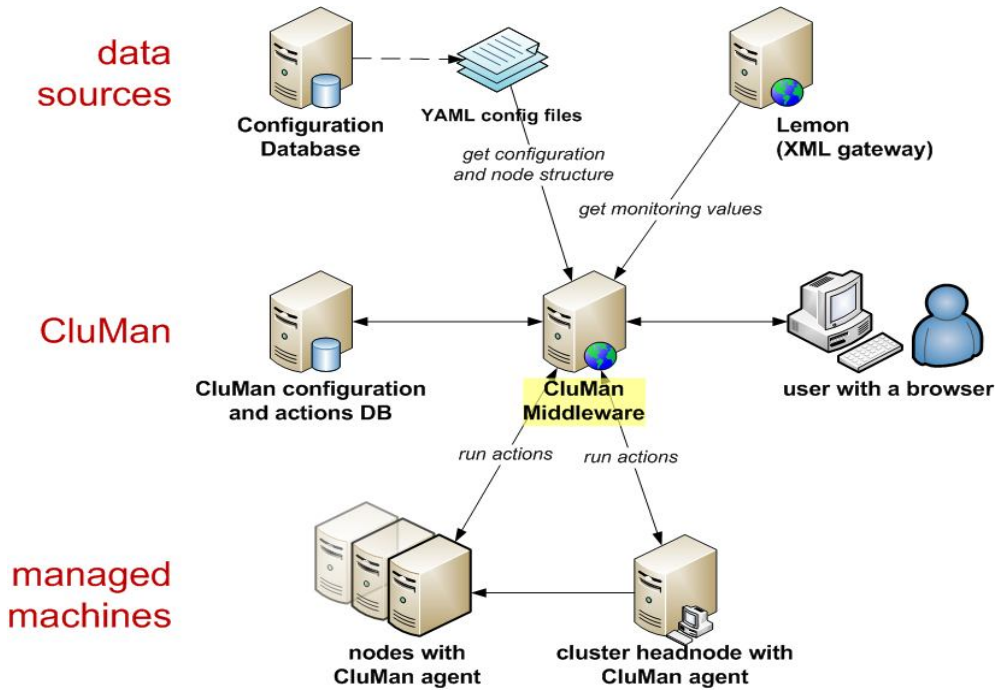
Cluster management in a very diverse environment with multitude of different clusters each to serve distinct purpose is a challenge. Primarily due to the fact of the sheer number of the available clusters and their sub-clusters and also due to the complexity of the individual services provided by them. Each service has their own procedures and scripts to correct problem that may arise in the course of everyday operation. While monitoring [12] and associated alarm systems [13] do handle means of reacting to the arisen problems, they do not provide proper means of detecting a deteriorating service.

With the help of high-density heat maps and special visualization techniques (automatically updated) to provide fast overview of user defined monitoring metrics we can address the requirement of the easy overview of the services and clusters. Adding possibility to select individual entities within a service or a cluster and launch a predefined or custom actions we can address issue of maintenance. Additional displays allow to follow progress of the launched actions and they allow also to see the result of the actions of the high-density heat maps.

Together with the proper authentication and authorization scheme, web interface and command line APIs, this is the ultimate tool for maintaining services at the highest level.

2. Design and architecture

CluMan is designed around the central database that stores configuration and information about actions being performed on the machines and the middleware server that performs scheduling, control and interface for the individual tools to CluMan database. The picture 1. shows the overview of the architecture.



Picture 1.

User connects to the CluMan service with a web browser or through the command line tools. After initial authentication and authorization he/she can define and execute actions for the services he/she is responsible for. The actions are then stored in the database and middleware server takes care of dispatching them to the individual machines.

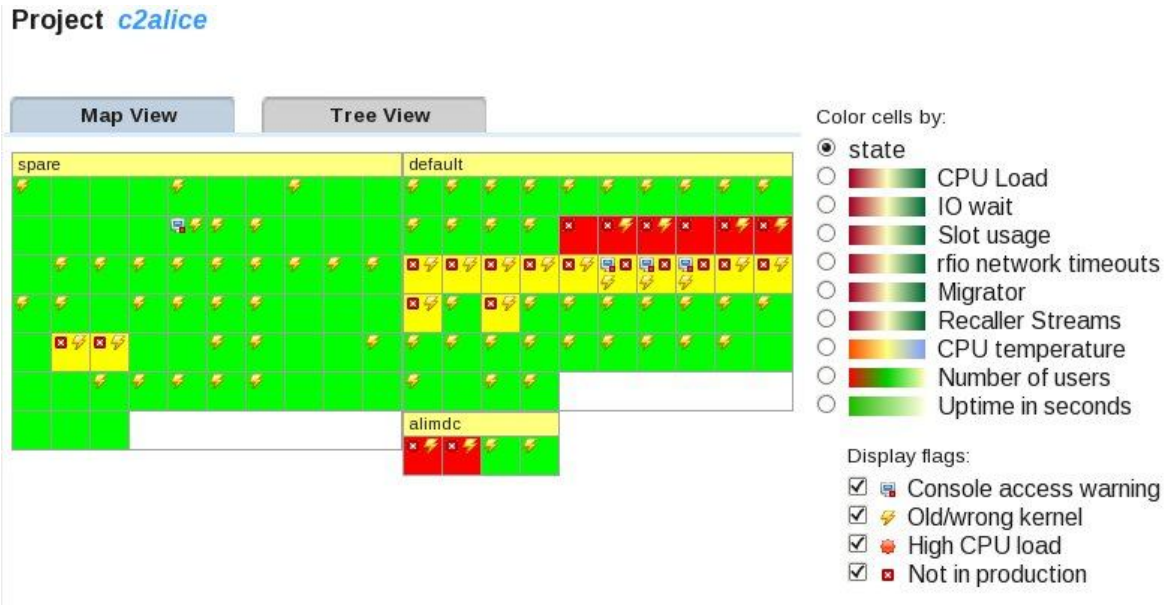
On the cluster nodes CluMan agent is responsible for executing the actions, following up their progress and in case of a need, performs maintenance and reports on the progress of the action.

2.1. Visualization

Proper visualization is the key element to CluMan. It focuses on allowing a user to choose aggregated views of the services according to his/her specification that suit him/her the best. It enables displaying anything starting from the state information, metric values of specific metrics in ranges, alarms or special flags as small pictures and/or other information in a colored way to allow easy identification of entities within service that may be out of the desired range.

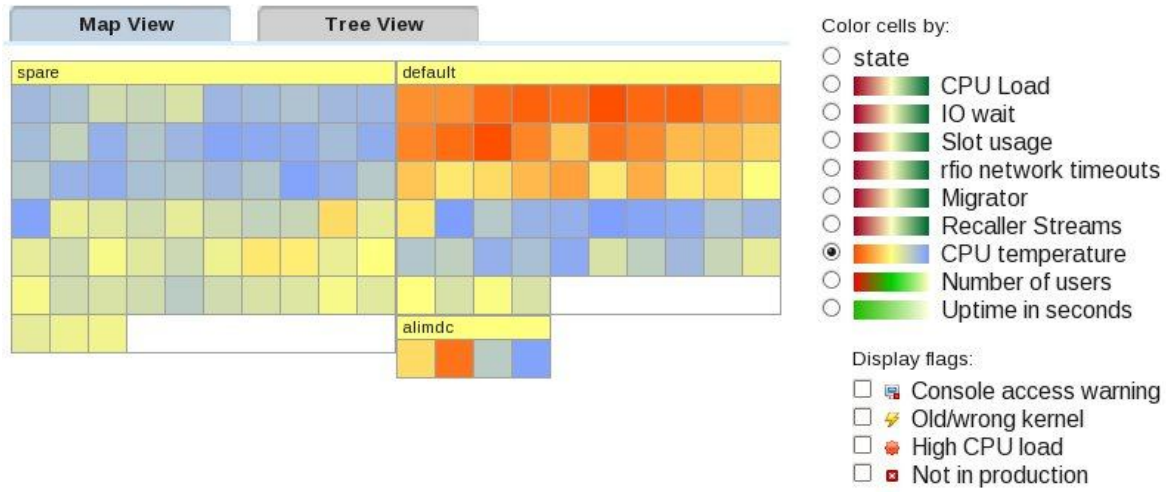
CluMan gets the metrics data from the monitoring system [12] through a well defined API (in case of CERN this is Lemon monitoring system). It requests and aggregates monitoring data for specific visualized metric on user request rather seamlessly.

An example of such heat-table is in Fig. 2. It displays a project composed of several sub-clusters as a grid. Individually defined states with their associated colors are displayed and additional flags are displayed as small pictures within the map. User can change the view based on his/her needs.



Picture 2.

Another example is in Fig. 3 where the project is displayed as a heat-map and given metric is displayed with predefined range of colors for the range of available predefined values of the metric. Apart from these displays, a tree view displaying project and its entities as a tree is available.



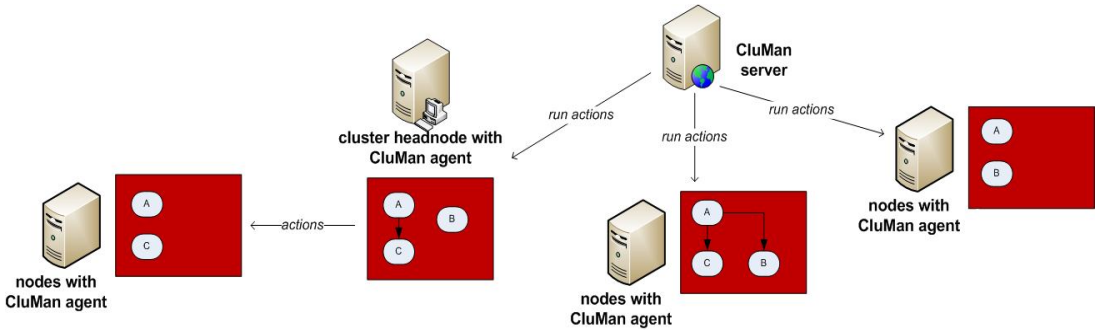
2.2. Actions

Actions are a must for the cluster management. The requirement on the actions is as follows:

- are defined scripts by service managers
- are executed by the CluMan Agent on the machines or the reference machines (head nodes) for the defined entities
- can be composed of chain of actions, dependent or independent of each other
- can be run sequentially or in parallel

- can be run on multiple entities in parallel with defined maximum level of concurrency
- chained action represent a simple DAG-like workflows with simple conditions – i.e. whether the previous action was successful
- progress is reported back and visualized on the web interface
- restart or abort of an action is possible
- a timeout for actions can be specified

CluMan middleware follows all of the actions, resolves possible issues with concurrency and timeouts of the actions, schedules further actions and closes the actions. It communicates with the CluMan agent on the node and through the agent schedules these actions in corresponding order.
An example of individual use cases is displayed in Fig. 3.



Picure 3.

2.3 Configuration

Individual project configuration can be imported into CluMan through the API in middleware that converts input YAML configuration files into database. Similarly, it is also possible for users to define the projects directly through the web interface.
Configuration consists of two parts – project entities and visualization. Project entities configuration file defines the whole tree structure of the project, clusters, sub-clusters and possibly more. It links also project entities into monitored entities – as these do not necessarily match either by their names or how they are registered in either of the systems. This also allows using different monitoring systems.
Visualization configuration file binds entities and monitoring metrics. It foresees different types of visualization of the metrics – using either flags, states or properties – each is visualized differently.
Actions can be configured by the individual action configuration files or through the web interface.

A sample configuration file follows – project entities with the tree structure:

```
# the entity hierarchy is defined here
entities:
        # parents must be defined before children
        -       name: c2atlas
                type: cluster
                description: castor2 atlas
                lemon_entities:
                        - { name: c2atlas, type: cluster }
                        - { name: c2atlas_srv, type: serviceclass }
        -       name: c2atlas_data
                # if parent element  is missing, then it's a root entity
                parent: c2atlas
```

```
        type: stager
        # description is optional
        description: c2atlas data stager
        lemon_entities:
            - { name: c2atlas%c2atlas_data, type: subcluster }
            - { name: c2atlas.c2atlas_data, type: stager }
    - name: lxfsra0801
      type: host
      parent: c2atlas_data
      description: c2atlas_data host
      lemon_entities:
            - { name: lxfsra0801, type: host }
```

Visualization file example:

```
# project configuration
details:
 # STATE
    - type: state
      entity_type: [ host, fileserver }
      metric: RemoteInfo
      field: EntityState
      states:
            - { value: production, caption: Production, color: #00ff00 }
            - { value: maintenance, caption: Maintenance, color: #ff0000 }
            - { value: standby, caption: Standby, color: #ffff00 }
 # FLAGS
    - type: flag
      name: Warranty expires soon
      entity_type: host
      metric: WarrantyInfo
      field: ExpiresSoon
      condition: = 1  # cond like “!= 2.6.9-67.0.15.EL.cernsmp”, “> 0.5” etc.
      icon: flag-WarrantyExpires.gif  # name of the icon file
 # PROPERTIES
    - type: property
      name: CPU Load
      entity_type: host
      metric: CPULoad
      field: Load
```

3. Implementation and choice of technologies

Choices have been made to reuse industry standard products [1-11] while allowing us to maintain license related issues to minimum. Middleware is based on a stateless event-driven architecture with a relational database backend. The API is provided in the form of the lightweight web services (REST), which is backed by the even-driven message queuing server, which asynchronously processes requests and connects to the underlying CluMan agents. Entire middleware stores its resource states in the relational database, e.g. Oracle, MySQL, PosgresSQL, etc. The REST API was developed within a python-based web framework called Django that allowed us to prototype rapidly and also to keep choice of databases behind CluMan open. It automatically provides administration interface, allows fast implementation of the interfaces for the individual components, supports cache systems and much

more. For the web visualization, we have opted for AJAX libraries to create more interactive system and allow users better experience.

4. Conclusion

The overall requirements and architecture of the CluMan system were described. The particular use cases and their motivation were shown in detail. The system is being developed at CERN and an open-source release of the system is planned with pluggable modules for different databases and monitoring systems.

References

[1] RedHat Cluster Suite, http://www.redhat.com/cluster_suite
[2] MOSIX, http://www.mosix.org/
[3] IBM Cluster Management System, http://h20311.www2.hp.com/HPC/cache/412128-0-0-0-121.html
[4] Aspen Beowulf Clusters, http://www.aspsys.com/software/abc/screenshots/manage/
[5] CMAv3, http://www.streamline-computing.com/index.php?wcId=118&xwcId=
[6] ParaStation Cluster Management, http://www.par-tec.com/ccc2.php?lang=en&page=parastation
[7] Windows Clusters, http://www.microsoft.com/windowsserver2003/enterprise/clustering.mspx
[8] Scali Platform Manager, http://www.scali.com/Products/platform-manager
[9] Ganeti cluster management, http://code.google.com/p/ganeti/
[10] ROCKS Cluster management, http://www.rocksclusters.org/
[11] VA Linux Cluster Management Software, http://vacm.sourceforge.net/
[12] Lemon monitoring system – http://cern.ch/lemon
[13] LAS (Lemon Alarm System) – http://cern.ch/lemon