Powerfarm: a power and emergency management thread-based software tool for the ATLAS Napoli Tier2

Alessandra Doria¹, Gianpaolo Carlino¹, Salvatore Iengo², Leonardo Merola¹², Sergio Ricciardi², Mariacarla Staffa²

¹ – INFN Napoli, Complesso Universitario di Monte Sant'Angelo, Napoli, Italy
² – Università degli Studi di Napoli “Federico II”, Complesso Universitario di Monte Sant’Angelo, Napoli, Italy

Corresponding author: Alessandra.Doria@na.infn.it

Abstract. The large computing power and the storage systems available in a Grid site or in a computing center are composed of several servers and devices, each with its own specific role in the center. A management and fault recovery system is of primary importance for management operations and to preserve the integrity of the systems in case of emergencies, such as power outages or temperature peaks. We developed Powerfarm, a customizable thread-based software system that monitors several parameters such as, for example, the status of power supplies, room and CPU temperatures and it promptly reacts to values out of range with the appropriate actions. Powerfarm enforces hardware and software dependencies between devices and it is able to switch them on/off in the particular order induced by the dependencies, so it’s also useful for the site administrator to speed up the management of scheduled downtimes.

1. Introduction

This work has been developed for the management of a Grid computing center, where a large number of devices are organized in clusters and interconnected together in order to reach common scientific objectives. In Grid terminology, a computing farm is a collection of computing and storage resources belonging to an administrative domain. A crucial factor of the success of these farms is the possibility to set up clusters made of heterogeneous nodes, each node with its own role in the farm. Besides the great advantages deriving from the possibility to set up clusters with different devices, the heterogeneity may also represent a challenging problem in the manageability of the farm. Each device – computing element, disk server, switch, router, or whatever else – has, in fact, its own hardware and software features that must be taken into account when the device must be turned on/off. Moreover, each node interacts with the others according to its logical role in the computing farm, creating dependency relations among the nodes. This complexity should be simplified as much as possible, especially in emergency situations such as power loss or temperature out peaks and in maintenance operations, in which a part of the farm is not working or some devices have to be turned off.

At the state of the art, several power management packages and tools are available [1, 2, 3], but none of them totally provides the necessary support and flexibility needed for Grid and data center infrastructures; all tools have been specifically designed to cope only with proprietary hardware, thus
lacking a common centralized system to manage all the heterogeneous devices of the whole center. In such a context we developed Powerfarm, a common centralized framework that enables to manage, check the status and turn on/off virtually any kind of device in the farm, both manually and automatically, respecting the specific roles of the devices in the farm.

2. System deployment
Powefarm has been developed for the Scope Project [4] and successfully deployed and validated in the Grid cluster located at the Napoli division of INFN (National Institute of Nuclear Physics) that has an important role in the computing of ATLAS LHC experiment [5]. The ATLAS Computing Model [6] is Grid-based, with distinct roles for different facilities (Tier-0 facility at CERN, large regional Tier-1 centers, smaller Tier-2 facilities) organized in a hierarchical structure. Napoli Grid site is a Tier-2 facility, currently of the dimension of about 100 multi-processors/multi-core servers and more than 200 TB of storage space. Tier-2 facilities may take a range of significant roles in ATLAS such as providing calibration constants, simulation and analysis. Tier-2 facilities also provide analysis capacity for physics working groups and subgroups.

3. Architecture
Powerfarm is a platform independent software system written in Perl (main core module) and Java (web interface) that runs under Linux, although it may be simply ported on other platforms thanks to the flexibility of Perl and Java programming languages.

Powerfarm is made up of the following components:

1. the Core Module
2. the SNMP Trap Handle Module
3. the Plugin Abstraction Layer Module
4. the Command Line and Web Interfaces
5. the XML Configuration Files
6. the Logging Module
7. the Notification Module

Figure 1. The Powerfarm Architecture
The diagram in Fig. 1 shows the architecture of Powerfarm together with the main actors of the system that are examined in the following paragraphs.
3.1. The Core Module
The Core Module is responsible for the management of the devices by means of three functions: “soft
c power off”, “hard power off” and “power on”. Powerfarm must operate both in maintenance and in
emergency conditions and it is necessary to foresee that malfunctioning devices could not respond to
commands; that’s why there are two types of power off modalities to shut down the devices. The “soft
power off” shuts down the devices in the canonical way, for example via SSH in case of a Linux
server. Instead, the “hard power off” is used whether the “soft power off” has not worked fine, thus
shutting down devices by physically powering off machines, by disabling the power supply units. Both
power on and off actions are taken respecting the dependencies between devices.
By “dependency” we mean that turning on/off a device is subordinate to turning on/off another one.
For example, before switching off an array of disks we have to turn off the servers mounting the
 corresponding storage space and vice-versa.

The Core Module is a Perl script that can be run at command line, executed through a Web
interface or activated by the SNMP Trap Handle Module.

3.2. The SNMP Trap Handle Module
The SNMP Trap Handle Module listens and captures the SNMP Traps [7] coming from the controlled
devices. Each trap corresponds to a particular event in the device and the Trap Handle Module informs
the Core Module of the detected events in order to take the opportune actions.

SNMP traps can be generated by different sources. In particular for our computing center:

1. The power supply is protected by a battery backup (UPS), in order to have a short time to
properly shut down the protected equipment in case the main power source fails. The UPS is
provided of a network control module, the UPS monitor [8], which sends SNMP traps when
changes in the power supply occur. When a power loss occurs (e.g. UPS switched to battery
functioning from AC), the UPS monitor triggers Powerfarm Core module to run with the
instruction to shutdown everything that is directly connected to the UPS and the other devices
that depend on the involved ones. When the power supply resumes, the traps from the UPS
monitor can activate the instructions to turn everything on automatically. This option is not
enabled in our deployment, since we prefer decide when to switch on the center, manually
with Powerfarm.

2. The racks holding the devices contain a control module, the Chassis Management Controller
(CMC) [14] that is equipped with sensors measuring a large number values, like temperature,
smoke/fire alerts, humidity, cooling system parameters, current in the power supply units and
more. The CMC sends SNMP traps when these values are out of range. For example, if the
temperature in a rack reaches a critical value, Powerfarm Core module may be executed with
the instruction to shutdown all the devices belonging to the rack; nevertheless, when the
temperature will reach a normal value again, Powerfarm can be executed to turn on again the
devices in the rack.

3.3. Plugin Abstraction Layer Module
Powerfarm can preserve the hardware and the software functionalities by checking status and
managing servers, worker nodes, storage resources, routers and virtually any other device. As the
cluster devices may profoundly differ from one another, Powerfarm manages them by means of
specific plugins that know the particular devices they manage and thus are able to interact with them
via standard or proprietary tools. The use of plugins creates an abstraction layer between the
Powerfarm logic and the heterogeneous devices available in computing farms, incorporating the
heterogeneity in small pieces of software written to manage only a particular category of devices.
Powerfarm uses plugins both to turn on/off the farm devices and to check their status. Powerfarm may
run plugins written in any programming or scripting language, that have to respect the return value
specification: a return value of 0 means that the plugin has successfully done its task; a return value different from 0 means that the plugin was not able to accomplish its task, and returns the error code with an optional description string. The Plugin Abstraction Layer Module provides a library of plugins for the most common devices, e.g. SSH, Intelligent Platform Management Interface (IPMI) [10], Baseboard Management Controller (BMC), Dell Remote Administration Controller (DRAC) [11], Power Distribution Unit (PDU) & Chassis Management Controller (CMC) [3], and make Powerfarm extendible by simply writing the appropriate plugins for the particular devices to control. For each device, the following plugins are defined:

1. Power on.
2. Soft power off.
3. Hard power off.
4. Check on.
5. Check off.

The “power on” plugin is used to turn on the device. If the device will not turn on within the set timeout, its lock on attribute is evaluated in order to decide if the process can go on or not. The “soft power off” is used as first attempt to turn off the device. If the device will not turn off using this plugin within the specified timeout, Powerfarm will run the “hard power off” in order to shut down the device at lower level, for example disabling the power supply unit (e.g. PDU) to the device (other methods of powering off are IPMI, BMC, DRAC, etc.). The last two plugins are used for checking the status on/off of devices.

3.4. The Command Line and the Web Interface

Powerfarm can be used by the system administrators for maintenance operations of the computing center. The various parameters - with which Powerfarm can be executed - let the administrator to select the devices to be managed with different levels of granularity: the whole Grid farm, all the devices belonging to a specific group (for example, all the worker nodes), a list of heterogeneous devices, everything is attached to the UPS, the devices inside a specific rack, and so on. Powerfarm exhibits a double interface: it can be executed by a command line interface or by its intuitive web interface (e.g. normally by the administrator for maintenance operations). Its XML configuration files can be edited either by the web interface or manually: the web interface will load always the latest version of the configuration files.

Figure 2 shows the aspect of the web interface, which is based on a JSP web server. At top level there are five main sections with which the user can:

- check the status of the farm;
- add a new device to be monitored or edit an existing one;
- search and process devices: the various search criteria allow user to perform exhaustive and precise selections of the devices to process;
- check the status of the power on/off operations currently active;

Figure 2. Powerfarm web interface: quick device selection
configure the settings of the application (e.g. reload a manually modified XML configuration file).

A typical life-cycle of interaction consists mainly in three operations:
1. select a group of devices to be processed;
2. once the list of the devices to be processed has been defined, proceed with the power on or power off operations;
3. make sure that all the involved devices have been normally processed checking the generated log.

3.5. The Set of XML Configuration Files
Powerfarm makes use of simple and intuitive XML configuration files in order to describe the structure of the site and the physical and logical dependencies between the devices. Powerfarm uses the following XML configuration files:
1. devices.xml, describing the managed devices associated with the corresponding plugins;
2. dependencies.xml, describing the dependencies among the devices (see section 4);
3. links.xml (optional), for the integration with other monitoring tools, like for example Nagios and Ganglia [12, 13].

3.6. The Logging Module
Powerfarm keeps track of all the events and activities that take place and logs them all with the dedicated Logging Module. The Logging Module shows all the information with a temporal timestamp in order to be able to fully reconstruct the events. Power outtakes, temperature peaks, automatically or manually turning on/off of the devices are all logged together with the precise information about the involved devices and their status.

3.7. The Notification Module
The Notification Module is in charge of notifying the system administrators in case any anomalous condition should happen, while Powerfarm performs the opportune actions. Notifications are sent via e-mail and/or SMS whether the appropriate SMS module is available.

4. Model and Dependencies
From a logical point of view, each server has its own role in a Grid farm, performing a specific service in the site. Some servers cannot perform their task if some other are turned off, in this sense we say there is a logical dependency among the servers. On another side, a server connected to an array of external disks will be stalled if the external storage device is suddenly turned off. In this case we speak of physical dependency. Therefore each device has its own logical and physical dependencies from other devices, inducing a specific chain of dependencies and thus a specific order to turn on/off the system. As an example, it could be necessary to turn off the storage elements only after that the last worker node has been turned off because there could be some job running on the worker node that is making use of some storage resources on the storage element. Nevertheless, it could be necessary to turn on the servers on which storage export areas reside before turning on the devices that will mount the storage area.

4.1. Graph of dependencies
Powerfarm represents dependencies among devices with the graph of dependencies, that is a directed acyclic graph (DAG) whose edges represent dependencies and nodes represent managed devices. As turn on and off dependencies represent the same (symmetrical) information, we have chosen to
represent only the turn off dependencies as direct edges in the DAG; the power-on dependencies can be simply obtained by reversing the edges directions. An edge between node$_1$ and the node$_2$ means that in order to turn node$_1$ off it is necessary to turn node$_2$ off before; similarly, in order to turn node$_2$ on it is necessary to turn node$_1$ on before.

In the model, each device can have whatever dependency with other devices, the only rule that has to be respected is that the dependencies do not form a cycle in the graph. This is a quite obvious constraint, as it would create an illogical cycle in the turning on/off dependencies chain. An example of the dependencies graph for a Grid farm is illustrated in Fig. 3 where SE stands for Storage Element, CE Computing Element, DS Disk Server, WN Worker Node, SS Storage System. As an example, it is represented that the disk servers may be turned on/off at the same time, as well as the worker nodes. Note that the SS$_2$ is used by both DS$_1$ and DS$_n$; this situation is correctly allowed in the DAG representation of dependencies.

Obviously the **power-off priority** – which is the order in which devices have to be turned off – follows the reverse direction with respect to power-off dependencies, crossing the edges backwards, thus starting the power-off procedure from the involved nodes without outgoing edges; note that as the graph is acyclic, there will surely be leaf-nodes.

Similarly, the **power-on priority** – which is the order in which devices have to be turned on – follows the reverse direction with respect to power-on dependencies, crossing the edges forwards, thus starting the power-on procedure from the nodes without incoming edges; note that as the graph is acyclic, there will surely be root-like nodes.

The dependencies graph is an explicit representation of the physical and logical dependencies and shows the turning on/off order and priority between the managed devices. The first part of the graph represents the dependencies that come from the physical constraints of how the devices are attached among them and to the power supply (UPS). The second part of the graph represents the software dependencies that come from the logical constraints of the devices, such as the role that each device plays in the farm. Anyway, there is no distinction in Powerfarm between the two kinds of dependencies that are treated at the same manner: they are just dependencies that must be followed to identify the devices that have to be turned on or off and their order.

The turn on/off procedure can be executed on a single element or on a set of elements in the graph; more specifically it can be requested on the following objects:

- a single device;
- a type of devices (e.g. all worker nodes, all storage elements, etc.);

![Dependencies graph for a grid farm.](image)
• a list of devices (e.g. “WN1, WN2, CE1, SE2”);
• a whole farm;
• everything that is connected to a device (e.g. everything connected to UPS1).

4.2. Layered tree of dependencies
When the power on/off procedure is executed on devices, graph dependencies have to be resolved into a layered tree structure in which the elements at each level can be turned on/off simultaneously, thus identifying a common layer of execution called dependency layer: at the same layer there are devices that do not have dependencies among them. The layered tree is constructed by breadth-first visiting the graph starting from the nodes on which the procedure is executed and by crossing the graph edges forward for power off, backward for power on, backward and then forward for recursive power on procedure. At each step of the breadth-first visit, new layers are added on the bottom or on the top of the last layer created, respectively for power on or power off procedures. As the starting graph is a DAG, it is always possible to obtain the layered tree of dependencies and, as the layered tree is made up of trees, each node in the structure has a single parent node, thus assuring that the whole process has a start and an end. Note that the dependency layers may also have different kind of devices at the same layer: this is a normal behavior that optimizes the process by parallelizing operations on devices that can be immediately processed. Once the layered tree of dependencies has been built, the power on/off procedure can start turning on/off the elements of the layered tree top down for power on, bottom up for power off. The dependencies trees obtained for the three function modalities – power on, recursive power on and power off – are represented in Paragraph 4.3.

4.3. Functioning modalities
Powerfarm has three main functioning modalities that are power on, recursive power on, and power off. The three functioning modalities execute their respective tasks on the specified devices respecting the physical and logical dependencies and optimizing the execution times by using parallel threads whenever it is possible.

4.4. Power on/off Algorithms
The pseudo-codes of the power on/off algorithms are shown in Table 1.

<table>
<thead>
<tr>
<th>Power_on (Node_list L, Boolean recursiveMode)</th>
<th>Power_off (Node_list L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Starting from the dependencies graph, build the layered tree of dependencies for the power on of the list of nodes L according to the recursiveMode;</td>
<td>1. Starting from the dependencies graph, build the layered tree of dependencies for the power off of the list of nodes L;</td>
</tr>
<tr>
<td>2. For each layer of the tree of dependencies, starting from the first one, do:</td>
<td>2. For each layer of the tree of dependencies, starting from the last one, do:</td>
</tr>
<tr>
<td>3. Check_on_Thread (current layer devices)</td>
<td>3. Check_off_Thread (current layer devices)</td>
</tr>
<tr>
<td>4. Power_on_thread (powered off devices)</td>
<td>4. Soft_power_off_thread (powered on devices)</td>
</tr>
<tr>
<td>5. While(timeout)</td>
<td>5. While(timeout)</td>
</tr>
<tr>
<td>6. Check_on_thread (still powered off devices)</td>
<td>6. Check_off_thread (still powered on devices)</td>
</tr>
<tr>
<td>7. If there is at least one device still powered off whose lock on attribute is set to 1, notify and exit; else, continue from the next layer</td>
<td>7. Hard_off_thread (still powered on devices)</td>
</tr>
<tr>
<td></td>
<td>8. If there is at least one device still powered on whose lock off attribute is set to 1, notify and exit; else, continue from the next layer</td>
</tr>
</tbody>
</table>

Table 1. Power on and power off algorithms pseudo-codes.
The “Power on” algorithm starts building the layered tree of dependencies according to the recursive mode specified. Then, starting from the first layer of the tree (i.e. layer 0), the status of each device is checked. Thus, the respective power on plugins are executed only on the devices that have resulted to be turned off (some device may be already on). Now, the status of the devices is cyclically checked until the last timeout is expired. Afterwards, if all the current devices are successfully turned on, the cycle can proceed to the next layer. Instead, if there is at least one device still off after the timeout, whose lock on attribute is set to 1, then the procedure exits and the administrator is notified.

The “Power off” algorithm is quite similar to the “power on”. Analogously, it starts building the layered tree of dependencies. Then, starting from the last layer of the tree (i.e. layer N), the status of each device is checked. Thus, the respective power off plugins are executed on the devices that have resulted to be turned on (some device may be already off, so they do not need to be turned off). Now, the status of the devices is cyclically checked until the last timeout is expired. Afterwards, if there is any device that has not turned off, the “hard power off” plugin is executed on such device. Now, if all the current devices are successfully turned off, the cycle can proceed to the next layer. Instead, if there is at least one device still on after the “hard power off”, whose lock off attribute is set to 1, then the procedure exits and the administrator is notified.

Note that each device at the same layer is checked and turned on or off simultaneously by means of parallel threads. The use of threads greatly reduces the working time of Powerfarm by parallelizing the operations on the devices that can be processed at the same time. This is a quite important aspect, especially when Powerfarm is performing a power off due to a power loss: in that case the autonomy time could be very limited, and so the working time of Powerfarm is reduced to the minimum.

4.5. Reversed modality

The “reversed modality” of Powerfarm is a particular option that can be very useful in case the situation that caused the turning off/on is reversed before the operation is completed and the cluster can be restored to its original state. As an example, if a power loss does occur, Powerfarm is automatically executed by the corresponding monitoring system (e.g. the UPS monitoring software) and starts to turn off all the involved devices. Now, if the power comes back while the power off procedure is being executed, the shutting down procedure can be interrupted and the current instance of Powerfarm can be inverted in order to start to turn on the devices that were turned off till that moment. Analogously, the reversed modality may invert the farm turning on if an event such as a power loss or a temperature peak does occur. These actions are totally configurable and priority emergency management mechanism has been implemented.

5. Multiple Instances and Emergency Management

We manage several instances of Powerfarm (that may be simultaneously launched) with a priority scheduling queue that ensures both the fairness and the emergency management of the different instances. Emergency instances of Powerfarm have in fact higher priority than regular ones – for example, an instance of Powerfarm triggered by a high temperature alarm has higher priority than a manually triggered one run for maintenance purposes.

If an emergency instance of Powerfarm was activated by a monitoring system, this instance will go directly to the top of the queue, interrupting any other running instance and it will be executed immediately, in order to promptly manage the emergency situations that has occurred. Nevertheless, administrators as well as automatic scheduling processes can operate simultaneously on the computing farm. As the farm is the same, multiple instances having normal priority are inserted in a queue. The instances will be executed in the order with which they have arrived, according to FIFO scheduling policy.

There is also the possibility for an administrator to cancel one instance while it is in the queue or, as we have seen, to invert its working modality (from on to off or vice versa). The working frame of Powerfarm for an AC power loss is illustrated in Fig. 3.
6. Conclusions
As we have seen, three situations can activate the execution of Powerfarm: (1) a change in the power supply (e.g. UPS switched from AC to battery functioning); (2) some sensors reporting out-of-range values (e.g. an anomalous temperature peak, a smoke/fire alert, etc.); (3) a manual operation by the system administrator for maintenance. Multiple instances, emergency management, several functioning modalities, double interface, XML configuration files and SNMP functionalities give to Powerfarm a great degree of flexibility. The abstraction layer implemented by the use of plugins enables Powerfarm to manage the different kind of devices that can be encountered in Grid and data center infrastructures. Besides, the execution time of Powerfarm has been dramatically reduced by using threads to perform parallel operations on set of independent nodes. This feature is of critical importance when it is necessary to turn on/off a set of nodes with the same dependencies (e.g. shutting down all the worker nodes) especially when the available time to accomplish the task is very low. To make an example, our center composed of about fifty servers and about 100 TB of disk storage can be completely switched off in about 5 minutes, following the described dependency chain. This time is compatible with the UPS battery duration in case of power loss. The time needed to switch on the center is about 10 minutes, supposed that everything had correctly received shutdown by Powerfarm itself. The time would be much longer in case some devices had not received a clean shutdown (for example a file system check could be needed) and in case of a manual switch on that would not respect the proper dependencies. Powerfarm may become an indispensable tool for emergency management of the modern Grid and data center infrastructures.

Figure 3. The Powerfarm working frame for an AC power loss
References