

# An Approach for Ensuring Reliable Functioning of a Supercomputer Based on a Formal Model

Alexander Antonov, Dmitry Nikitenko, Pavel Shvets<sup>(✉)</sup>, Sergey Sobolev,  
Konstantin Stefanov, Vadim Voevodin<sup>(✉)</sup>, Vladimir Voevodin,  
and Sergey Zhumatiy

Research Computing Center, Lomonosov Moscow State University, Moscow, Russia  
{asa,dan,shpavel,sergeys,cstef,vadim,voevodin,serg}@parallel.ru

**Abstract.** In this article we describe the Octotron project intended to ensure reliability and sustainability of a supercomputer. Octotron is based on a formal model of computing system that describes system components and their interconnections in graph form. The model determines relations between data describing current supercomputer state (monitoring data) under which all components are functioning properly. Relations are given in form of rules, with the input of real monitoring data. If these relations are violated, Octotron registers the presence of abnormal situation and performs one of the predefined actions: notification of system administrators, logging, disabling or restarting faulty hardware or software components, etc. This paper describes the general structure of the model, augmented with details of its realization and evaluation at supercomputing center in Moscow State University.

**Keywords:** Reliability · Autonomous operating · Model of supercomputer · Monitoring · Supercomputing

## 1 Introduction

A supercomputing center maintenance practice determines a set of strict requirements for the technologies and facilities supporting supercomputer operation: maintaining high productivity of the supercomputer, constant monitoring of all potential sources of emergency situations and the performance of all critical components, automatic decision-making by maintenance and support systems, and guaranteed operator notification about the supercomputers current status, among other requirements. Until all of these requirements are met, neither the efficient operation of the supercomputer nor the safety of its hardware can be guaranteed.

The reasons for such requirements are clear. Supercomputers are expensive and therefore downtime is unallowable. The demand for supercomputers is high, and that means the maximum operational equipment should be available to the users. Supercomputers require high power consumption, which means their status needs to be monitored closely to avoid an equipment loss. And the complexity of the maintenance problem grows extremely fast with the vast number

of components in any modern supercomputer. Fulfillment of these requirements sets the foundation for the Octotron system. Lets make a formal supercomputer operation model and make it available to the Octotron as an input data. The Octotron will constantly observe the current state of supercomputer using the monitoring systems and compare it to the model. If actual supercomputer state does not correspond with the model, Octotron can perform one of the preprogrammed actions, such as notifying the administrator via SMS, disabling the malfunctioning device, and so on. With this approach we, first, guarantee complete control over the situation; second, guarantee compliance between the expected and actual behavior of the supercomputer; and third, will be confident that we will find out about any event that deserves our attention.

This approach leads to a number of useful outcomes. In particular, it allows us to not only guarantee the reliable operation of the existing fleet of systems within a supercomputing center, but also to ensure continuity in maintenance when moving to a new generation of machines. Indeed, once an emergency situation arises, it is reflected in the model along with the causes and traceable features, and an adequate reaction is programmed into the model.

The paper is organized as follows. In Sect. 2 we describe interesting existing solutions for the discussed and related problems. In Sect. 3 we briefly state what our goals for the Octotron system being created are. Section 4 contains detailed description of the structure of our system, with the focus on one of its most important parts supercomputer model. This structure description is continued in Sect. 5 with the explanation how our system operates. In Sect. 6 we show how the Octotron system is being used on real supercomputer systems in our university. Finally, Sect. 7 contains conclusions and acknowledgments.

## 2 Background and Related Works

The work to ensure reliable supercomputer operations has been going on for a while, and a broad range of materials and methods has been accumulated. The following approaches to handling emergency situations are provided [1]: forecasting potential failures and their consequences; preventing failures; reducing the number of errors and their impact on system health; ensuring resilience against failures.

In global practices, the resilience for a supercomputer is primarily viewed in the context of ensuring reliable application execution. The methods available to support the execution of a large application in a potentially unreliable environment are based on creating checkpoints and logging communications, which allows the application to recover from failures in computing system components [2]. However, these mechanisms do not address equipment safety issues. Maintaining reliable operations and monitoring the status of a supercomputing system can be done using proprietary vendor hardware and software solutions (HP BTO Software [3], xCAT [4]). An alternative option is to install and configure one of the freely distributed monitoring systems (Nagios [5], Zabbix [6], Ganglia [7]) and writing a set of scripts to respond to specific subsets of potentially dangerous situations.

Currently, the aforementioned systems do not use a coherent model or description of a computer as input data. In our opinion, lack of model usage makes very difficult, if not impossible, to analyze and react properly on complex global fault situations, which concern not only one component such as one node or server, but a variety of cluster components. The powerful Zenoss [8] monitoring system offers the target system modeling concept, but that is only understood as automatic identification of the system configuration. Failure detection based on a system of rules has been implemented in ClustrX Safe/AESS (Automated Notification/Equipment Shutdown System) [9] by the T-Platforms company. However, this system is focused solely on infrastructure monitoring and does not affect the supercomputers computational and software parts. It is also the vendors proprietary solution.

Another interesting recent development is the Iaso [10] system designed by NUDT University in China. This system supports the autonomous operation of the Tianhe-2 supercomputer, the current leader of the Top500 rating. It is an integrated piece of software that addresses all of the issues of automatic fault detection and elimination within system components, locating the root causes of failures, performing self-diagnostics and self-testing of the computing system, and recovering applications after failures. The ideas implemented in Iaso partially correlate to the ideas of this project. Even though Iaso is declared to be a universal software complex, it requires a modified Linux kernel on the clusters computing nodes with its own client modules installed. Iaso receives a description of the target system as input, but this is mainly used for monitoring and controlling the network infrastructure. Iaso is not publicly available at the moment.

Therefore, currently there have not been found any open system that uses formal model for maintaining reliability of a supercomputer.

It should be mentioned that Octotron system is intended to work with existing monitoring systems like collectd, Nagios, or Zabbix as data source thus avoiding unnecessary duplication of data collecting agents.

### 3 Requirements for Octotron System

The primary functional requirement for the Octotron system has in fact been formulated: the system must allow a supercomputer to independently control its own operation by comparing its current state to a predefined model. Since this task is not a trivial one, and Octotron must operate in a complex supercomputer environment, an additional set of requirements has been formulated which the system must meet:

1. be able to control all key failure causes in a supercomputers hardware and software components;
2. allow the monitored area to be expanded to include any components that were not originally controlled by the system but caused a failure;
3. be able to react to emergencies independently from the operator by performing a set of predefined actions to eliminate a failure or to notify support team;

4. support the current generation of teraflops and petaflops supercomputers and be ready to work with the next generation of supercomputing systems, where the number of components will increase by an order of magnitude;
5. verify the integrity of its own operation and evaluate the adequacy of information it has received on the state of the supercomputer;
6. accumulate experience from previous supercomputing system support, minimizing the number of repeated failures on both existing and prospective supercomputers.

Considering these requirements, an architecture was proposed for the Octotron system based on a formal model of supercomputer operation, which is described in the next section of this paper.

## 4 The Supercomputer Model

Octotron represents the supercomputer model in the form of a graph [11] (Fig. 1), which is used to describe all typical modern supercomputer components and relations between them. The model is accompanied by a set of rules and reactions. The rules help the supercomputer to register a failure or emergency, and the reactions describe what actions need to be taken once a rule is triggered.

Vertices in the graph correspond to physical or logical components of the supercomputer that need to be monitored: computing nodes, UPS modules, job queues, software components, licenses, etc. The criteria are simple: everything that the efficient operation of a supercomputer depends on must be reflected in the model. The graph edges correspond to the relationships between components, e.g. consists of, provides power to, connected with Infiniband. Each vertex in the graph is associated with a set of attributes which describe that components status: processor temperature, amount of memory, number of jobs in a queue, etc. The Octotron system updates attribute values through the supercomputers own monitoring systems (like collectd or Nagios) or directly via external interfaces on the components. The latter method is used, in particular, to work with engineering infrastructure over the SNMP protocol, or to interact with a GSM modem, or to get the current status of software licenses.

The core of Octotron is written in Java programming language. The system generates operative graph in the memory, while using Neo4j database as a long-term storage. All write requests are executed on both graphs so that we can keep current supercomputer state description up-to-date, while read requests use only memory-stored graph for improved performance. Neo4j can be used independently from the Octotron system, serving as a standardized interface for side tools, such as visualization, analysis, debugging and so on. Database support is optional and can be disabled, but in case of termination or failures all data will be lost.

Python language was chosen as the primary language for model description. We use the Jython interpreter, which executes the code on a Java virtual machine and allows classes from a Java code to be used in a Python program. Since Python is a rather simple and clear language, even an untrained person can

create a model, following the examples and documentation for the Octotron system. Heres a sample of code describing the model shown on Fig. 1.

```
# creating basic components
room = CreateObject()
chiller = CreateObject()

# creating components with attributes
ups = CreateObject({"sensor" : {"load" : Long()}})
air_cond = CreateObject({"sensor" : {"fluid_temp" : Long()}})
hot_aisle = CreateObject({"sensor" : {"air_temp" : Long()}})
rack = CreateObjects(3, {"sensor" : {"temp" : Long()}})

# creating contains edges
OneToOne(room, ups, "contains")
OneToOne(room, air_cond, "contains")
OneToOne(room, hot_aisle, "contains")
OneToEvery(room, rack, "contains")

# creating power edges
OneToOne(ups, air_cond, "power")
OneToEvery(ups, rack, "power")

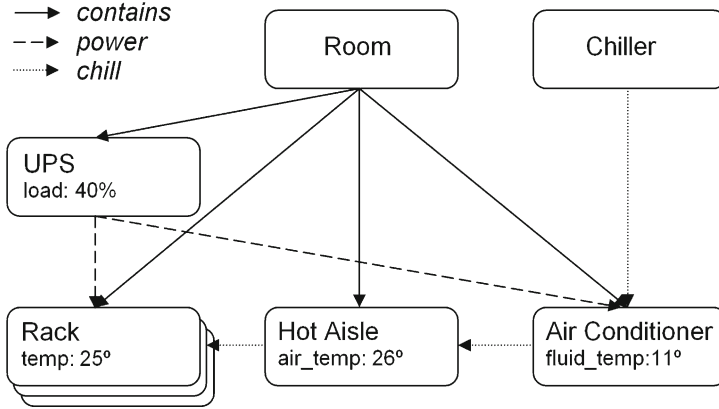
# creating chill edges
OneToOne(chiller, air_cond, "chill")
OneToOne(air_cond, hot_aisle, "chill")
OneToEvery(hot_aisle, rack, "chill")
```

Once created, such model can be rather easily updated in case supercomputer structure is needed to be changed (for example, computational core upgrade or some topology modifications is going to be made) by simply modifying this model description and running model creation process again. In this case all attributes from the older version of the model will persist if its object still exists in the new model.

## 5 Octotron System Operation

After a supercomputing system model has been created, we need, first, to add actual data on the structure of supercomputer components; second, to organize verification of abnormal statuses for the components (i.e. rules); and third, to define the respective system reactions to these situations. All of these activities are implemented in the Octotron system.

The supercomputer model is accompanied with a set of rules, which define deviations in the supercomputers behavior from what is set in the model. Each attribute in the model is linked to a set of rules that is triggered with every change in the value of the given attribute. In particular, the rules can be used



**Fig. 1.** General idea of the Octotron supercomputer model: vertices (computing nodes, UPS modules, job queues, software components, licenses, etc.), edges (relationships), attributes

to determine the rate at which attribute values change, which helps control situations similar to this: a rapid growth in the error rate on network interfaces can signal about a network failure. The rules are assigned to vertices and edges but can access attributes not only from the same vertex but also from adjacent vertices with the given relationship type.

Most commonly used templates of rules and their compositions available in our system are the following:

1. compare an attribute value with a constant or another attribute. Example: identify when a component temperature exceeds predefined limits;
2. aggregate values of several attributes and check the result. Example: identify when temperature exceeds the predefined limits for several sensors in a hot aisle;
3. check attribute values in two adjacent (according to the graph) vertices. Example: different operational modes on a pair of ports directly connected to an Ethernet network;
4. convert an attribute value for a further usage with other rule. Example: convert an absolute value to percentage value;

In addition, reaction modifiers allow extending checks in two ways:

1. check if reaction condition is maintained over a certain period of time. Example: LoadAVG value (the number of OS processes ready for execution) at a node can awhile go beyond ordinary limits, but if the value stays high for a long time, this can indicate problems on the node;
2. check if a reaction condition repeats several times in a row. Example: a node fails an SSH access check more than three times in a row.

If a rule registers an abnormal operation of a component, the corresponding reaction is triggered. A reaction represents a set of actions, such as recording

data in the log file, notifying system administrators by e-mail or SMS, or invoking a custom command. Rules and reactions, as well as the model itself, are written in Python using a special library.

Set of rules to check and report number of available nodes is shown below. It writes a message to a log file if more than 20 % of nodes are unavailable (ping failed).

```
def NodeGroupModule(nodes_count):
    return {
        "var" : {
            # calculate the number of neighborhood objects with
            # attribute "ping" equals to "false"
            "failed_ping" : ASoftMatchCount(False,
                EDependencyType.OUT, "ping"),

            # converts the number to a percentage value,
            # basing on provided total nodes count
            "failed_pct_ping" : ToPct("failed_ping",
                nodes_count),

            # returns true if the percentage is below 20 or
            # false otherwise
            "failed_pct_ping_ok" :
                UpperThreshold("failed_pct_ping", 20),
        },

        "react" : {
            # reaction for an object is triggered when the
            # attribute "failed_pct_ping_ok" becomes "false"
            Equals("failed_pct_ping_ok", False):

                # reaction writes the message with category
                # "Danger" to a log file
                ( Danger("tag", "SYSTEM").Msg("descr",
                    "too many unavailable nodes")

                # when the "failed_pct_ping_ok" attribute turns
                # back to "true" -- another entry will be added to
                # a log file
                , Recover("tag", "SYSTEM").Msg("descr",
                    "nodes are available again"))

        }
    }
```

Data on a supercomputers status is imported into Octotron from external sources: a monitoring system or directly via component interfaces. Octotron is not tied to any specific monitoring system, but can work with any one through a custom import module. Data from various supercomputer subsystems can be

imported into Octotron at various intervals, depending on the necessary reaction speed to failures in each respective component.

Users and administrators can interact with the system with a web browser through HTTP requests. All requests are divided into few categories: view requests, modification requests and control requests, with an option of a separate authentication. View requests allow users to query all information about objects, attributes, rules and reactions. Modification requests allow to import data and change the model. Control requests are used by administrators to perform such actions as terminating model, suppressing all reactions and accessing system performance metrics.

Data import into Octotron system is made by import modules a set of tools that convert output of different monitoring system or methods to required format. Import modules do not have a strict structure; the only requirement for module is to provide a valid import request. We are planning to provide a default modules for typical monitoring tools and methods, such as SNMP, collectd, ping and ssh checks, written on Bash language and Python.

The Octotron system also features several levels of built-in diagnostics and self-diagnostics. To verify the appropriate operation of the system, an independent service graph with basic needed functionality is added to the system database. Octotron processes the specific rule for this service graph that involves this functionality, which in turn is verified by the external process. If the check passes, the system is considered to be operating correctly.

Another check is related to the frequency by which attribute values associated with the actual supercomputer components are updated. Each sensor attribute can be assigned a timeout, during which its value should be received from the source. If the value is not updated within that time, it indicates that either the respective component is faulty, or there is an error in its monitoring status. Similarly a timeout can be specified during which the value must change or, on the contrary, remain stable.

The last verification level is the monitoring of one Octotron system with another Octotron system: two independent copies of Octotron that are monitoring two supercomputers are being monitored by the third one. This third instance provides two types of diagnostics: (1) checks if Octotron processes are alive and (2) send requests to service graph described earlier to verify that it is handled correctly.

## 6 Evaluation of the Octotron System at Moscow State University

Octotron system is deployed on two Moscow State University supercomputers: Chebyshev (60 TFlops performance peak, 625 nodes, 5,000 cores, 42 racks and 1 hot aisle) and Lomonosov (1.7 PFlops peak, 5,000 nodes, 82,000 cores, 115 racks and 5 hot aisles). Chebyshev supercomputers model contains 10,228 vertices, 24,698 edges and 205,044 attributes. Lomonosov supercomputers model contains about 116,000 vertices, 332,000 edges and 2,400,000 attributes. Models



reflect the following supercomputer components: a power supply system (UPS, battery modules); a cooling system (chillers, in-row air conditioners, environment monitoring); a management component (access nodes, job queues); a computing component (chassis, nodes, disks, memory); a shared file system; Ethernet network (switches, ports); and an Infiniband network (switches, network manager). The following relationships exist between the said components: contains, chill, connected via Ethernet, connected via service network, connected via Infiniband network, includes and provides power to.

As an example, we describe how Octotron operates on the Chebyshev supercomputer mentioned above. The main supplier of operational data for this supercomputers computing components is the monitoring system based on collectd. The hardware infrastructure supplies data via SNMP.

Every 10 min, the Chebyshev supercomputer hosts report the following data: the number of active SSH user sessions; the number of active software licenses; the number of jobs in each of the six partitions of the supercomputer (total number, jobs queued, jobs being executed, completed jobs); the number of processors (total, available for job execution, blocked); and the GSM modem account balance for sending emergency SMS notifications.

The following data is collected from all computing nodes at a higher frequency (every minute): the temperature inside the node; each processors temperature; ID of a job being executed; the status of the file system; memory status (total/free/occupied memory); Infiniband card status (sent/received packet counter, errors); Ethernet card status (errors); average node load, number of zombie processes, and other system data. SMART information on HDD status is additionally collected from nodes equipped with local hard disks.

The following data is collected from the shared file system every 10 min: free/occupied space, the performance, as well as the status and load of each blade module. Data from Ethernet switches are collected at the same intervals.

Information on the supercomputers climate control system is collected every minute. It includes data from several indoor temperature and humidity sensors and the status of each of 8 in-row air conditioners (air/coolant temperature before/after the air conditioner, and a number of various alerts). More than 60 parameters are recorded at the same interval from each of the five UPS units: the status of grid power and the UPS operating module, battery status, etc.

About 160 rules are used to control the operation of the Chebyshev supercomputer. Some of them are: GSM modem account balance is close to the deactivation limit; failures in the operation of two or three chillers; substantial increase in the error rate on network interfaces; number of user sessions at the host is below threshold.

In both cases of Lomonosov and Chebyshev supercomputers, overhead of monitoring systems is very low: <1 % of computation load of nodes and <1 % of communication network bandwidth is used. Octotron instance for Lomonosov supercomputer works on one dedicated server (2x Xeon E5450, 32 GB RAM), uses <10 % of CPU and 6 GB of physical memory (20 GB of virtual). Chebyshev instance requires much less resources.

Within the evaluation period Octotron system is being used together with ClustrX Safe/AESS (mentioned in Sect. 2) on Lomonosov supercomputer. In this case Octotron only notifies about found failures but do not perform automatic actions like shutting down hardware. We still had few opportunities to fully check functionality of our system, since in most cases failures are local and not very hard to find. The most common failures are: node went down; temperature of HW component is rising; IB card failed; etc. A few less common cases that were discovered: not expected RAM memory volume on several nodes; chiller failure; not enough disk space on utility server; password attack on utility server. One interesting case we'd like to mention here. During summer outside temperature was getting very high, and at the same time small part of cooling hardware was down. Supercomputer became to overheat, and Octotron system notified administrators that this temperature problem is getting critical within the whole machine.

## 7 Conclusion

Currently, Octotron is used to support the reliable autonomous operation of the Chebyshev and Lomonosov supercomputer at MSU. In the near future, it will be deployed on the new Lomonosov-2 supercomputer.

The architecture and implementation of the Octotron system meets all of the criteria developed earlier. Potential sources of supercomputer failure are identified, thanks to a complete model of the target system and automatic monitoring of each component. Octotron is easily scalable for any supercomputer with petaflops performance, as confirmed by early experiments with the Lomonosov supercomputer. There are interesting areas for development, too. For example, further scalability can be achieved by breaking down the supercomputer model into a set of smaller models and launching several independent copies of Octotron, each monitoring its own part of the system. The overheads of the current version of Octotron are small and have no serious impact on the supercomputers performance.

Now, the Octotron system is following several directions of development. One is related to developing a shared bank of potential failures. Furthermore, interactive model visualization tools are being developed which allow the model not only to be viewed, but also promptly modified. Automated model creating tools will be expanded as well. Another interesting and promising area is an in-depth analysis of the flow of events taking place inside a supercomputer. This is aimed both at locating the root cause of failures and emergencies, and at forecasting such situations in the future. It is important that all of this functionality can be implemented by developing new modules and linking them to the already fully functioning Octotron kernel.

It is also noteworthy that the supercomputer model gets a value of its own as the key repository of knowledge on its structure. Moreover, the supercomputer model can be effective for educational purposes, since it can easily be used to demonstrate key components of a supercomputer and the relationships between them.

Octotron is available under an open MIT license [12, 13].

**Acknowledgments.** This material is based upon work supported by the Ministry of Education and Science of the Russian Federation (Agreement N14.607.21.0006, unique identifier RFMEFI60714X0006).

## References

1. Snir, M., Wisniewski, R.W., Abraham, J.A., Adve, S.V., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., Chien, A.A., Coteus, P., Debardeleben, N.A., Diniz, P., Engelmann, C., Erez, M., Fazzari, S., Geist, A., Gupta, R., Johnson, F., Krishnamoorthy, S., Leyffer, S., Liberty, D., Mitra, S., Munson, T.S., Schreiber, R., Stearley, J., Hensbergen, E.V.: Addressing failures in exascale computing. *Int. J. High Perform. Comput. Appl.* **28**(2), 129–173 (2014)
2. Cappello, F., Geist, F., Gropp, W., Kale, S., Kramer, B., Snir, M.: Toward exascale resilience: 2014 update. *Supercomput. Front. Innovations* **1**(1), 5–28 (2014)
3. HP, BTO Software (part of HP Software Division). <http://www8.hp.com/us/en/software/enterprise-software.html>
4. Ford, E., Elkin, B., Denham, S., Khoo, B., Bohnsack, M., Turcksin, C., Ferreira, L.: Building a Linux HPC Cluster with xCAT. An IBM Redbooks Publication. <http://www.redbooks.ibm.com/abstracts/sg246623.html?Open>
5. Nagios monitoring system description. <http://www.nagios.org/>
6. Zabbix system. <http://www.zabbix.com/>
7. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* **30**(7), 817–840 (2004)
8. Zenoss - Open Source Network Monitoring and Systems Management. <http://www.zenoss.org/>
9. Clustrx SAFE (AESS) software. <http://www.t-platforms.com/products/hpc/software/clustrxproductfamily/clustrx-safe.html>
10. Lu, K., Wang, X., Li, G., et al.: Iaso: an autonomous fault-tolerant management system for supercomputers. *Front. Comput. Sci.* **8**(3), 378–390 (2014)
11. Antonov, A.S., Vad, V., Voevodin V.I., Voevodin, S.A. Zhumatiy, D.A. Nikitenko, S.I. Sobolev, K.S. Stefanov, P.A. Shvets: Securing of reliable, efficient autonomous functioning of supercomputers: basic principles and system prototype. *Vestnik UGATU (Scientific Journal of Ufa State Aviation Technical University)* **18**(2(63)), 227–236 (2014)
12. Octotron core repository. [https://github.com/srcc-msu/octotron\\_core](https://github.com/srcc-msu/octotron_core)
13. Octotron framework: modeling and monitoring of complex computer systems. <https://github.com/srcc-msu/octotron>

Parallel Processing and Applied Mathematics  
11th International Conference, PPAM 2015, Krakow,  
Poland, September 6-9, 2015. Revised Selected  
Papers, Part I

Wyrzykowski, R.; Deelman, E.; Dongarra, J.; Karczewski,  
K.; Kitowski, J.; Kazimierz, W. (Eds.)

2016, XXIV, 622 p. 229 illus., Softcover

ISBN: 978-3-319-32148-6