Research on the Health Diagnosis Module of Large-scale Clusters

Cong Yang
Cloud Computing Research Center
Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences, Shenzhen, China
cong.yang@siat.ac.cn

Wen-long Du
Ideal Institute of Information and Technology
Northeast Normal University
Changchun, China
duwl623@nenu.edu.cn

***Abstract*—a large number of low-level performance metrics include process, virtual and physical machine metrics that can be measured to identify a node or even a cluster health status. Traditionally, nodes in the cluster are monitored and managers need to analyze each metrics and alarming messages from monitoring tools to identify the health status of clusters. However, this process would cost too much time on some insignificant metrics and with less efficient because most clusters have more than hundreds nodes and it's impossible for one manager to check too much metrics in each nodes. In this work, we demonstrate that more time can be saved by simplify metrics set, scoring each nodes and diagnosis nodes health status by decision tree. Specially, this work first experimentally verifies and sorts the degree of relation between node health and different metrics. After that, we collect and score the training set by load increase testing. Thirdly, we construct a decision tree by training set. Finally, a health diagnosis module is composed by previous process, algorithm and decision tree. We evaluate the Health Diagnosis Module (HDM) on the Normal PC cluster. Experiments show that HDM can precise diagnose nodes and clusters' health status with more than 89% accuracy rate.**

Keywords-cloud computing; decision tree; health diagnosis module;

I. INTRODUCTION

Cluster Health Diagnosis is used for cluster running errors detection, reporting and correcting. By collecting the basic metrics from each node of cluster, the analysis module will detect the physical condition of cluster and each node. With the help of cluster health diagnosis reports, a set of powerful functions such as changing the process running status and running process locking, etc. help to keep cluster healthy and stable. Traditional cluster health diagnosis methods are manual that managers use special commands or software connecting to the single node and checking up node running status. However, those approaches are lack of automation and intelligence. What's more, traditional monitoring tools have limited abilities of process and virtual layer metrics collection [1, 2, 3, and 4], automated errors analysis and mining process [2], etc. All of these drawbacks motivated us to design and develop a Cluster health Diagnosis Module for large-scale virtual and physical clusters.

This paper proposes a Health Diagnosis Module, named HDM. HDM is composed by three main components: (1) Metrics Training Set, which combined by healthy score and Load Increase Testing results. (2) Decision tree for diagnosis. (3) Score and node errors report.

We summarize our contributions as follows:

- We experimentally reveal an automatic cluster health diagnosis module over the incoming streams and storage metrics.
- We construct a new decision tree and architecture for metrics scoring and health diagnosis.

The rest of this paper is organized as follows: Section 2 presents the architecture, principle of diagnosis module and the process of decision tree construction and cluster health diagnosis. Evaluation results from simulations and testbed experiments are shown in Section 3. Section 4 concludes the paper.

II. PRINCIPLE OF DIAGNOSIS MODULE

In this section, we present HDM design and principle in details. Firstly, we discuss the HDM architecture. After that, we introduce the Decision Tree construction processes in details. The overall design of HDM is shown in Figure 1.

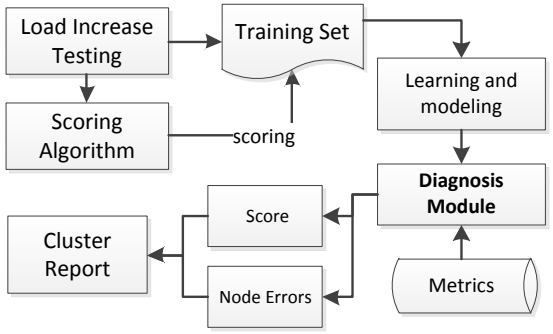

Figure 1. Architecture of HDM.

A. Overview of the arichitecture

As shown in Figure1, HDM is composed by three main components. Training Set is formed by Load Increase Testing

results and scoring results. By learning and modeling, Decision Tree will be constructed in the Diagnosis Module and waiting for new metrics data is collected and flowing in. New metrics will be diagnosed, scored and formed into the cluster report.

HDM is a non-stop thread running in the Database Service of Xmon [5], which helps users to manage nodes or clusters more effectively and conveniently by reading the diagnosis reports.

B. Cluster health diagnosis process

There are two main process of the cluster health diagnosis process. Firstly, through the load increase testing process to get a list of metric refers to the cluster and node health obviously and the relationship between these metrics and the health score. By this process, we get a Training Set with the health score of each metric group.

Secondly, by the method of machine learning and modeling, Decision Tree will be constructed. Working with the Score and node errors report module, Decision tree periodicity read the monitoring data from NoSQL Database and forming cluster health reports.

For these processes, main research points include Load Increase testing and Attribute selection, Training Set collecting and scoring, Decision Tree construction.

1) Feature selection and Load Increase testing. A system often contains a large number of low-level performance metrics that can be measured to identify a service status [6]. For example, Linux provides over 100 OS-level metrics; Intel CPU contains hardware performance counters for more than 20 parameters. Including too many attributes in a synopsis could be time complexity prohibitive and irrelevant attributes in a synopsis may even cause a loss of prediction accuracy [6]. Therefore, we use the compounded metrics include all metrics appeared in the command *vmstat*, *proc/cpuinfo, proc/diskstats, proc/iomem, proc/swaps, proc/vmstat*, a total of 35 monitoring common metrics.

Thereafter, we continually collect the 35-fetures during the Load Increase testing on service machines. For cloud platform, statistics process is divided into two parts: Virtual Machine Load Increase and Task Load Increase. We analyze the changes of 35-features during the two processes and select the top 10 largest rate of change features as Attribute metrics.

As shown in Table 1, in the process of Virtual Machine Load Increase, we prepared 3 different configurations of Virtual Machine: a CPU intensive, a memory intensive and an IO intensive. The testing program will load a virtual machine randomly, which simulate the real running environment more accurate.

TABLE I. VIRTUAL MACHINE LIST

ID	*VCPUS*	*Memory (max, KB)*	*NETS*	*VBDS*	*NETTX (max, KB)*	*NETRX (max, KB)*
1	1	2097000	1	3	97500	27900
2	2	1097000	1	1	97500	27900
3	1	2097000	1	2	975000	279000

The configuration of test server is as follows:
OS: SUSE Linux Enterprise Server 11 (x86_64)
CPU: 16 core GenuineIntel, Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
Memory: 6047744 kB
Swap: 8385920 kB

In the process of Task Load Increase, we use multi-MPI-task [7] to calculate the value of π .By the idea of Calculus, a 1/4 round is unlimited divided into n shares. Each sharp is closer to rectangle when n is infinite, and the width w is closer to $w = 1/n$, distance is $d(i)$. Therefore, the height of each sharp is $\sqrt{1-[d(i)]^2}$. Each MPI process stands for a share of n shares and running individually. Multi-MPI result will be summed together to get the value of π. In this case, the number of MPI process is based on the number of n and more number of n means more accurately of π calculates. The command of this MPI process is:

mpiexec -np 100 ./pi

In this command, 100 stand for the number of n . We increase 100 numbers of n each time and monitor the changes of 35-features in the same time, till n ups to 1600.

Through two processes of Increase testing previously, we obtain the average change rate of 35-features. After that, we sort these average rates and select top 10 largest change rates as Attribute features which shown in Table 2.

TABLE II. AVERAGE CHANGE RATES LIST (TOP10)

Feature	*Virtual Machine*	*Task*	*Percentage %*	*Order*
Incore Memory	0.007815786	0.004278983	0.31	10
Net In and Out	1.238635954	0.958545451	49.43	1
Total Process	0.010419492	0.025136915	0.42	7
Used Memory	0.009159214	0.005124115	0.37	9
CPU Load	0.468833289	0.622358512	18.71	2
CPU Usage	0.211479755	0.422222366	8.44	4
Free Memory	0.418507614	0.321555566	16.70	3
Buffered Memory	0.113782488	0.095844555	4.54	5
Cached Memory	0.017069217	0.032551225	0.68	6
CPU Frequency	0.010130371	0.010552254	0.40	8

2) Training Set collection and scoring. Final Score of clusters and nodes health status can help managers to quickly view and manage whole clusters. In HDM, score will be average divided into 10 levels from 0 to 100. Score 0 means the cluster and nodes is turned off or dead. When score is less than 10 or between 10~30 means there are some errors in the cluster waiting for fixed by manager. And when score is between 40 and 90 which means some nodes has warnings in the cluster. Score 100 means there is no virtual machine or task that is running on the cluster.

As shown in Table 1, different metric has different percentage of changing rate during the testing which means we could use this percentage for scoring each node. However, different service

has different hardware configuration which leads to the different cardinal point of each metrics. What's more, HDM module needs to fit different demands of different physical environment. For these reasons, 10-metrics will be calculated by proportion. For details: Free Memory should be calculated into the proportion of Total Memory as training property; Total process should be calculated into the proportion between the running process number and the total process number in the equilibrium environment; Net In and Net Out, CPU Frequency, One Minute CPU Load should be calculated into the proportion between current value and the average value of same metrics in the equilibrium environment, respectively; Percentage of CPU Usage can be used for scoring directly for its value will not change with the change of cardinal point. By the method previously, more than 300 metric groups are collected.

However, original data cannot use for Decision Tree construction because there are some errors and unusual noisy data. Therefore, we need to find and clean it by linear regression method [8]. For details, we detect the outliers by the method of Cluster Analysis in each metric type. [9] Figure 2 shows the Net metrics outliers analyze and we take the usual data (red ellipses) as Training Set metrics.

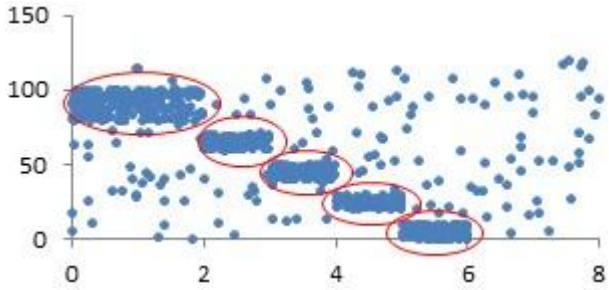

Figure 2. Net metrics outliers analyze.

After cleaning and integration, 200 clean metric groups are formed into Training Set, partly shown in Table 3.

3) Decision Tree construction. We declare the *Score* as Class Label in this 200-Metric Groups Training Set. As mentioned previously, *Score* is divided into 10 numbers from 0 to 100, that is: {10,20,30,40,50,60,70,80,90,100}, which means there are 10 different classifications($m=10$), marked as $\{c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8,c_9,c_{10}\}$. In this Training Set, $c_1,c_2.....c_9,c_{10}$ includes {8,7,8,9,6,12,14,11,14,11} samples, respectively. As in (1), we need to use this formula to calculate the Information Gain of each sample. Specially, p_i is the probility between any data object and classifcation C_i , caculated as $p_i = s_i / s$.

$$I(s_1,s_2,...,s_m) = -\sum_{i=1}^{m} p_i \log(p_i) \qquad (1)$$

Therefore,

$$I(s_1,s_2,...s_{10}) = I(8,7,8,9,6,12,14,11,14,11)$$
$$= 0.291+0.268+...0.350 = 3.270$$

In this paper, we are starting with the feature of *Net* as an example. As shown in Table 3, we need to count the distribution of *Net* samples in 10 *Score* levels by the formula of (2).

$$I(s_{1j},s_{2j},...,s_{mj}) = -\sum_{i=1}^{m} p_{ij} \log(p_{ij}) \qquad (2)$$

Followings are the distribution of *Net* samples.
For *Net*=:

$$0\sim2,\ s_{11}=0,s_{21}=0,\cdots s_{101}=9, I(s_{11},s_{21}...s_{101})=1.390$$
$$2\sim3, s_{12}=2,s_{22}=2,\cdots s_{102}=2, I(s_{12},s_{22}...s_{102})=1.386$$
$$3\sim5,\ s_{13}=2,s_{23}=3,\cdots s_{103}=0, I(S_{13},s_{23}...s_{103})=1.205$$
$$>5,\ s_{14}=3,s_{24}=3,\cdots s_{104}=0, I(s_{14},s_{24}...s_{104})=0.878$$

The next step is to calculate the Information Entropy of *Net* samples by the formula of (3).

TABLE III. TRAINING SET (PARTLY)

Net	*Process*	*CPU Load*	*CPU Frequency*	*CPU Usage*	*Used Memory*	*Free Memory*	*Buffered Memory * 100*	*Cached Memory*	*InCore Memory * 100*	*Score*
>5	>1.12	>4.5	0~0.9	>0.6	>0.98	>0.358	>0.16	0.38~0.9	0.2~0.25	10
3~5	0~0.92	2~4.5	>1.16	0.32~0.6	<0.42	>0.358	0.1~0.16	0.38~0.9	0.2~0.25	30
0~2	0.92~1.12	0~2	0.9~1.16	0.14~0.26	0.42~0.98	0~0.058	0.04~0.1	0.38~0.9	0.2~0.25	80
...	...	...	...	...	...	...	...	...	...	...
3~5	0.92~1.12	0~2	0.9~1.16	0.14~0.26	<0.42	0~0.358	0.04~0.1	0~0.38	>0.25	60
0~2	0.92~1.12	0~2	0.9~1.16	0.14~0.26	0.42~0.98	0~0.358	0.04~0.1	0~0.38	0.2~0.25	100
0~2	0.92~1.12	0~2	0.9~1.16	0.14~0.26	0.42~0.98	0~0.358	0.04~0.1	0.38~0.9	0~0.2	90
>5	>1.12	>4.5	0~0.9	>0.6	>0.98	>0.358	>0.16	0.38~0.9	0.2~0.25	10
3~5	0~0.92	2~4.5	>1.16	0.32~0.6	<0.42	>0.358	0.1~0.16	0.38~0.9	0.2~0.25	30
0~2	0.92~1.12	0~2	0.9~1.16	0.14~0.26	0.42~0.98	0~0.058	0.04~0.1	0.38~0.9	0.2~0.25	80

$$E(A)=\sum_{j=1}^{v}\frac{s_{1j}+s_{2j}+\cdots+s_{mj}}{s}I(s_{1j},\ldots,s_{mj}) \quad (3)$$

$$E(Net)=\frac{33}{100}I(s_{11},s_{21}\ldots s_{101})+\frac{27}{100}I(s_{12},s_{22}\ldots s_{102})$$
$$+\frac{23}{100}I(s_{13},s_{23}\ldots s_{103})+\frac{17}{100}I(s_{14},s_{24}\ldots s_{104})$$
$$=1.259859138\approx 1.260$$

Finally, we can get the Information Gain of Net samples by the formula (4).

$$Gain(A)=I(s_1,s_2,\cdots,s_m)-E(A) \quad (4)$$

$$Gain(Net)=I(s_1,s_2,\cdots s_{10})-E(Net)=2.010$$

We are using the similar process to get other samples' Information Gain as shown in Table 4.

In Table 4, we can see that property of *Net* has the biggest Gain, which means that Net should be the Root Node of Design Tree [10, 11], partly shown in Figure 3.

TABLE IV. INFORMATION GAIN LIST

Sample	***Gain***
Net	2.010
Total Process	1.023
CPU Load	1.430
CPU Frequency	0.559
CPU Usage	1.380
Used Memory	1.621
Free Memory	0.267
Buffered Memory	0.950
Cashed Memory	0.830
InCore Memory	0.076

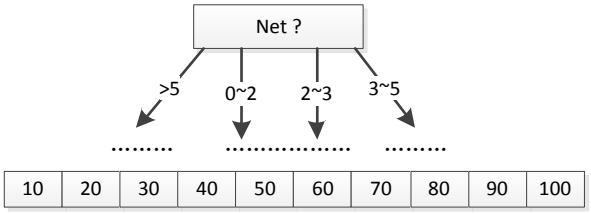

Figure 3. Design Tree of HDM (Partly).

4) Cluster health diagnosis and reporting. After the Design Tree is constructed, monitored metrics can be detected and scored. Total score of cluster is combined by the average of score of each node with $\log(s_i)$. The reason of using the process of logarithm is to strength the weight of low score on individual node. By the process of logarithm, when there are some errors on one single node, the total score of cluster will drops evidently. The final health reports contains two parts: total score of cluster and errors (warnings) list on each node.

III. EVALUATION

In this section, we present a quantitative analysis of HDM along with an account of experience gained through deployments on normal PC cluster. For the analysis, we measure the accuracy rate of HDM.

A. Experiment Setup

In this paper, we use normal PC cluster to evaluate HDM. The normal PC Cluster is Lenovo table PC Ubuntu cluster in the Cloud Computing Research Center in SIAT, currently be used for web-based system testing. The cluster consists of approximately 5 dual-processor SMP nodes. Each SMP is a Lenovo M5500 desktop, each of which contains AMD 2.70GHz Athlon (tm) 7750 CPU, 2.00GB memory and one 500GB disk. The cluster also includes a uses Extreme Networks Huawei switches for Gigabit I/O connectivity between the nodes. All nodes in the cluster run the Ubuntu 10.04 LTS-64 bit.

B. Experiment Results

Accuracy Rate shows whether HDM is suitable for cluster diagnosis. To get this result, we randomly manual input cases in the cluster, run HDM process and record every reports from HDM. The case list includes 6 different states of clusters as shown in Table 5.

TABLE V. CASE LIST

Case No.	***Case Description***
1	One node is dead in the cluster
2	One node's CPU Usage is more than 80%
3	One node's Net Flow is 9 times more than stable state
4	One node's Used Memory is more than 90%
5	Running 1000 MPI process on one node
6	Do nothing on cluster

After 100 times case testing, we can get 100 reports. We manual calculate the score of each case firstly and then check the mapping HDM reports to identify whether they are corresponding. We will count the number of correct reports and then we can get the accuracy rate of HDM. The testing result of HDM is shown in Figure 4.

From Figure 4, we can see that the lowest Accuracy Rate is 89% when the case testing number going to 45 times. The distribution of results is stable, approximately, random separated in the section of 88% to 100%.

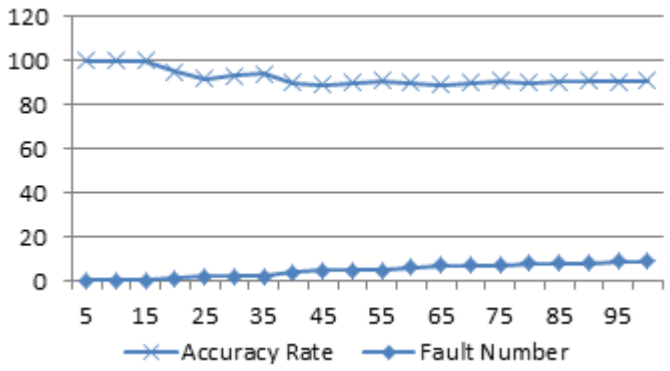

Figure 4. Accuracy Rate testing result

IV. CONCLUSION

In this paper, we presented the architecture, principle, and evaluation of HDM, Health Diagnosis Module of Large-scale Clusters. The main purpose of HDM is to help people manage cluster more convenient and efficient by automatic errors detection and reporting. By selected features and Design tree, HDM can quickly detect errors and remind managers. Case testing shows that HDM can diagnosis cluster with high accuracy rate.

Our future job will concentrate on HDM in specific SaaS cloud platform, like multimedia cloud platform, because different cloud platform has different framework and structure.

ACKNOWLEDGMENT

This material is based upon work supported by the Shenzhen Institutes of Advanced Technology (SIAT) and Northeast Normal University. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Chinese Academy of Sciences, or other funding parties.

We do appreciate the invaluable data and discussion offered by Cheng-Zhong Xu from Wayne State University and Jue-Hong from Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Science.

REFERENCES

[1] Matthew L.Massie,Brent N.Chun and David E.Culler, “The ganglia distributed monitoring system: design, implementation, and experience,”, Parallel Computing 30 (2004) , pp.817–840.

[2] Matthew J.Sottile and Ronald G.Minnich, “Supermon: A high-speed cluster monitoring system,”, Cluster Computing, 2002. Proceedings, pp. 39–46.

[3] R.Buyya,"PARMON: a portable and scalable monitoring system for clusters,", Software: Practice and Experience,vol.30, pp.723-739,2000.

[4] Z.Liang, Y Sun, and C Wang. "ClusterProbe An Open, Flexible and Scalable Cluster Monitoring Tool", In IEEE Intercational Workshop on Cluster Computing, pp. 261-268,1999.

[5] Cong Yang, Cheng-Zhong Xu, etc. “Xmon: A Lightweight Multilayer Open Monitoring Tool for Large-scale Virtual Clusters”.

[6] Jia Rao, Cheng-Zhong Xu. " Online Capacity Identification of Multitier Websites Using Hardware Performance Counters", In IEEE Transactions on Parallel and Distributed Systems, Vol.22, No.3, pp. 426-438,2011.

[7] “MPI”, http://www.mcs.anl.gov/research/projects/mpi/.

[8] G.H.John. Enhancements to the Data Mining Process. Ph.D.Thesis, Computer Science Dept.,Stanford University, 1997.

[9] E.Knorr and R.Ng.Algorithmes for mining distance-based outliers in large datasets. In Proc.1998 Int. Conf. Very Large Data Bases, pages 392~403, New York, NY, August 1998.

[10] C.E.Brodley and P.E.Utgo_.Multivariate versus univariate decision trees. In Technical Report 8, Department of Computer Science, Univ. of Massachusetts, 1992.

[11] D.E.Brown, V.Corruble, and C.L.Pittard. Acomparison of Decision tree classfiers with backpropagation neural networks for multimodal classi_cation problems. Pattern Rcognition, 26:953~961,1993.