February 6, 2007

RT0705

Computer Science 15 pages

Research Report

Failure Management for Large Scaled Systems

Hideki Tai and Takayuki Kushida

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Failure Management for Large Scaled Systems

Feb 6, 2007: Version 1.0

Hideki Tai, Tokyo Research Laboratory, IBM Research
Takayuki Kushida, Tokyo Research Laboratory, IBM Research

Abstract

Large scaled systems confront relatively frequent failures which degrade the productivity of the system. An occurrence of failure incurs job restart and administrator's work like problem determination. Job restart deteriorates substantial performance of the system, and increase of administrator's work will be reflected to maintenance cost. Further improvement in the productivity of large scaled systems depends on how failure is managed well. The goal of this study is to improve productivity of large scaled systems from the system management point of view, i.e. substantial performance and administration cost. Substantial performance is measured by average job slowdown and system utilization.

This document describes failure prevention as a usable technique to improve substantial performance, and an approach for problem determination of large scaled systems as a means to lower the administration cost. Failure prevention is to take proactive actions based on fault prediction and anomaly detection. This contributes to evade performance degradation caused by faults. The cost of problem determination is another impact of failures. This should be lowered by advanced tools that can provide informative information to administrators based on the data collected from the system. In addition, this document describes the data analysis platform which becomes a common infrastructure for the analyses required by failure prevention and problem determination tools.

1. Introduction

Large scaled systems confront relatively frequent failures which degrade the productivity of the system. MTBF (Mean Time between Failures) of large scaled systems are reported to be just only a few hours or tens of hours at most [1]. Further scaled out systems will confront shorter MTBF. An occurrence of failure incurs job restart and administrator's work like problem determination. Job restart deteriorates substantial performance which is measured by job throughput and utilization of the system. Increase of administrator's work will be reflected to maintenance cost. Therefore, further improvement in the productivity of large scaled systems depends on how failure is managed well. The goal of this study is to improve productivity of large scaled systems, i.e. substantial performance and management cost related to failures, by means of failure prevention and problem determination tool. This document will focus on describing the data analysis platform which is required to implement these techniques and mechanisms for large scaled systems.

Failure prevention is one of the techniques to evade performance degradation caused by failures. It can be achieved by taking proactive actions towards predicted occurrence of faults or critical events. A simple example of proactive action is to detach a node when it is predicted to become faulty in the near future. The node may be attached to the system again when the node is predicted that it will not become faulty for the present. More advanced means for failure prevention are proposed in several literatures. Failure prevention mechanisms are based on fault/event prediction which can be achieved by analyzing event logs or time-series data like CPU load, temperatures, and fan speeds. These analyses need to be performed continuously in a real-time manner at high throughput to realize failure prevention for large scaled systems.

Problem determination is one of the most time-consuming human tasks incurred by failures. Administrators need to inquire historical data and event logs to find the root cause of a failure occurred to the system. This work will become more difficult and intricate as the system scales more and analysis data increases. So, administrators need advanced tools that pre-process and analyze the large amount of data from various points of view to indicate useful information for finding the root cause. Although advanced data analysis on large amount of data tends to take longer time, they should respond immediately because they are invoked in an interactive manner from a tool by the administrators.

Both failure prevention and problem determination are based on data analyses on large number of historical event logs and resource status data. In this study, we will develop a data analysis platform which enables high throughput analysis in a soft real-time manner for on-line analyses, and immediacy for interactive analyses in a best-effort manner. These properties are essential to next generation failure management systems for large scaled systems. The data analysis platform schedules the executions of analyses so that it meets the requirement of as many analysis tasks as

possible with limited computing resource provided for failure management. When the analysis server is overloaded, the data analysis platform drops some analyses tasks which are judged to be unimportant or duplicated so that it can execute more important analysis tasks. Importance of each analysis task is determined based on the current status of the objective resource (e.g. node,) type of the analysis (on-line or interactive,) elapsed time since the last analysis was performed on the objective resource.

In the following sections, related works are described in section 0. Failure prevention and a tool for problem determination are described in section 3 as the envisioned applications of the data analysis platform. Detail of the data analysis platform is described in section 4. Finally, outline is summarized in section 5.

2. Related Work

Several techniques and algorithms for failure prevention are proposed in the literatures. One of the proposed actions is fault aware job scheduler. Oliner et al. [8] proposes fault aware job scheduling algorithms that should improve system performance. According to their simulations based on actual job history and failure data collected from a large-scale computer, their algorithms showed noteworthy improvements in average job slowdown and system utilization. Average job slowdown is improved even with 10% of fault prediction confidence. System utilization is improved as the confidence of fault prediction increases.

Another proposed action for failure prevention is controlling job migration and checkpointing. Li and Lan [9] proposes a mechanism that optimally chooses migration, checkpointing or no action to reduce the execution time. Like the fault aware job scheduling proposed in [8], their mechanism uses fault detection. According to their experiment based on a real failure log captured from a supercomputer, their algorithm showed steady improvement in the mean execution time compared to traditional checkpointing/restart strategy.

Every failure prevention mechanism depends on fault prediction or anomaly detection. Sahoo et al. [6] proposes techniques and methods to predict critical events. They evaluated time-series analyses, rule-based classification algorithms, and Bayesian network models using event logs and system performance logs collected from a system consists of 350 nodes. The results suggest that rule-based classification algorithms can be used to predict critical rare events up to 70% accuracy at 400 seconds of time window before predictions. Brandt et al. [5] proposes a monitoring and analysis tool incorporated with a Bayesian inference scheme to indicate aberrant nodes. As an example, they applied the method to find aberrant node whose temperature is different significantly from others. Such aberration can be considered as an early indicator of future problems.

Failure management systems need to monitor event logs and system status like temperatures, fan speeds, and voltages. The captured data can be filtered and alerted on each managed node, or they can be collected and stored in a central location so that it can be used by various purposes like displaying the system status, retrieving RAS (reliability, availability, and serviceability) information about the system [2], finding the root cause of a failure, and data analyses such as anomaly detection [5] and failure/event prediction [6].

Monitoring systems like RMC (Resource Monitoring and Control) of RSCT/CSM, Ganglia [2], and Supermon [3] provide scalable monitoring function. However, they are mainly used only to present the system status on per-node basis, or to alert events based on a fixed threshold. Monitoring systems should be integrated with on-line data analyses to implement proactive fault management. The integration portion controls analyses and monitoring behaviors to achieve scalable analysis and low perturbation for the monitored nodes.

3. Applications of the Data Analysis Platform

3.1. Failure Prevention

Failure prevention is to take proactive actions to avoid job interruption based on fault prediction or anomaly detection. Although all failures cannot be prevented without perfect fault prediction, possibility of job interruptions can be lowered by taking appropriate actions based on existing anomaly detection or fault prediction techniques. Several techniques have been proposed, but further research is still needed to make them widely adopted.

One of the required researches is the runtime mechanism for fault predictions and anomaly detections. The runtime enables to deploy fault prediction and anomaly detection mechanisms to large scaled systems by several functions for scalable on-line analysis: data collection, data management, and analysis scheduling. The data collection function should achieve low perturbation so that the system performance will not be degraded because of the failure prevention. The data management function manages the life cycle of the collected data, and it provides a data access method required for on-line analysis. The analysis scheduling function is required to perform various kinds of analysis on large number of resources.

One important requirement of the analyses for failure prevention is soft real-time. Analysis must be completed before when the system can take proactive actions to prevent failures. Otherwise, the analysis result becomes useless for proactive fault management. This seems easy if the system can provide a powerful analysis server which is capable of complete every analysis request within a fixed delay. However, this will not be feasible because large scaled systems have numerous resources to be monitored and the total analysis workload exceeds the capacity of an analysis server. Another requirement is to achieve as higher analysis throughput as possible.

Suppose that a system consists of 10,000 nodes, analyzers monitors 10 variables per node, and analysis is performed on each variable at intervals of 10 minutes (600 seconds,) each analysis is given only 6 mili-seconds. This is too short to perform various data analyses for fault predictions or anomaly detections. When we implemented anomaly detection algorithms derived from [11], a simple algorithm took 20-40 mili-seconds and another complex algorithm took about 1,500 mili-seconds.

Moreover, we cannot assume that analysis workload is fixed to a specific amount. Some additional analyses may be programmed to filter false alarms emitted by preceding lightweight-but-sensitive analysis. Sometimes the analysis workload will exceed the capacity of an analysis server if too many false positive alarms are emitted simultaneously. The runtime should manage the execution of analyses to solve these problems.

3.2. Problem Determination

When a failure occurred to a system, administrators are required to inquire historical data to find

the root cause of the failure. However, historical data increases as the system scales, and it becomes too intricate to extract useful or important information from large amount of data. This will be reflected to management cost of the system. Administrators should have advanced tools that extract useful information by pre-processing and analyzing the historical data.

Tools are required to respond as fast as possible because they are used in an interactive manner. On the other hand, analyzers of problem determination are expected to take much longer time as the system becomes larger and the number of subject data increases. The data analysis runtime should provide a parallel execution mechanism for this type of analyses aiming to achieve short response time.

4. Data Analysis Platform

The data analysis platform provides an infrastructure to collect, store, and perform on-line analyses on resource status data of a large scaled system. As we described in section 0, the data analysis platform should achieve soft real-time and high analysis throughput for on-line data analyses, and it should also achieve short analysis response time for interactive data analyses. In addition, the data analysis platform should be scalable so that it can host large number of analyses on time-series data collected from a large scaled system. It schedules the executions of analyses and controls multiple analysis servers. It also controls the activity of sensors on each node to minimize the monitoring overhead. Analysis results, e.g. failure predictions and anomaly detections, are expected to be used for proactive fault management.

4.1. Architecture

Figure 1 describes the overall architecture of the data analysis platform. It consists of a master server and a set of analysis servers each of which takes charge of hundreds or thousands of managed nodes. The master server runs a database management system (DBMS) and provides database connections to the analysis servers. There are three kinds of databases on the master server. The first database is used for storing configuration of the analysis servers. The second database is used for storing collected resource status data. The third database is used for storing analysis results. All analysis servers share the same configuration so that they collect the same types of data and perform the same analyses. The master server is responsible for dealing with failures occur to the analysis servers. The master server watches the analysis servers and replaces faulty analysis server with a backup server if available. If no backup server is available, the master server tries to add the managed nodes under the faulty analysis server to another managed node if possible. Otherwise, the master server updates the status of the managed nodes to unknown, and leaves the managed nodes as they stand until any analysis server becomes available for them.

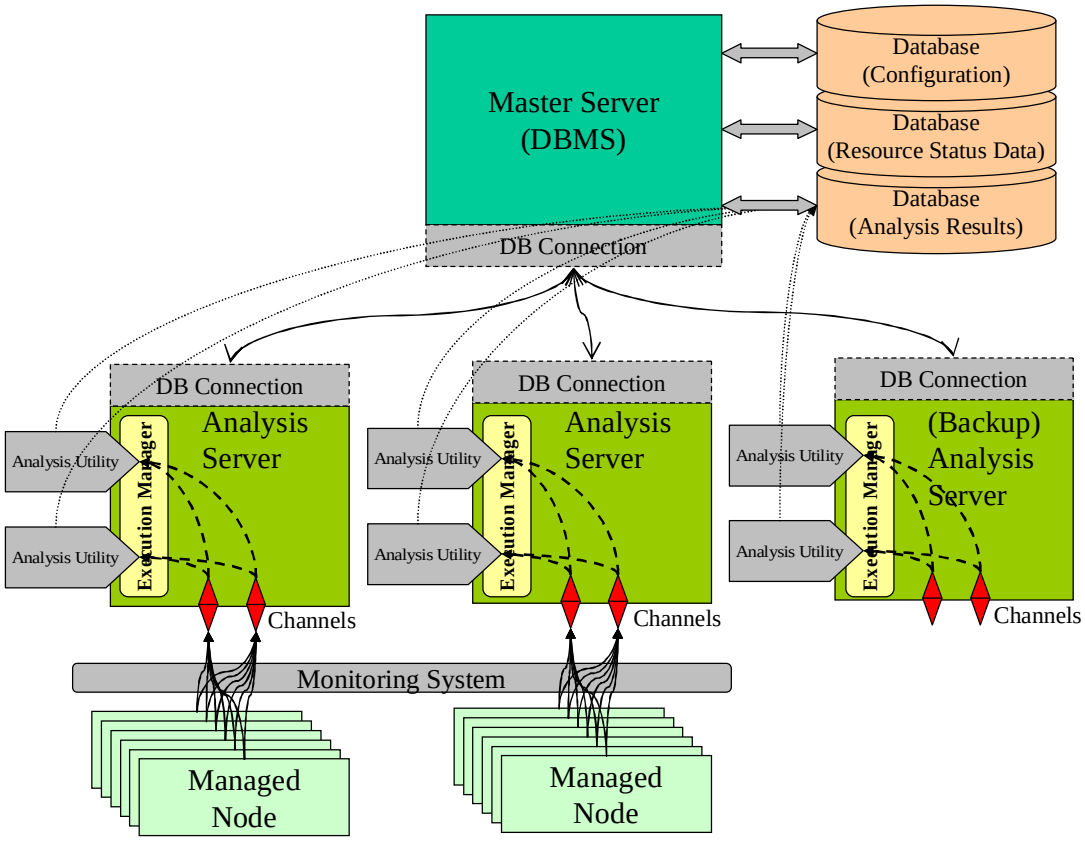

Figure 1: Overall Architecture of the Data Analysis Platform

4.2. Analysis Utility

Analysis utility is the unit of a data analysis module like fault prediction and anomaly detection. Several analysis utilities can be plugged into an analysis server. The data analysis platform provides a programming model and a framework for the analysis utilities which are explained in section 4.5 below. An analysis utility consists of analysis triggers and analyzers. An analysis trigger is responsible for filtering data, accumulating data, and triggering an analysis execution. It listens to one or more channels and accumulates data required for performing an analysis. The accumulated data is packed as an analysis task object, and put into the queue of the execution manager. The analysis utility receives the analysis result when the execution manager finished the analysis, and stores the analysis result to the database.

4.3. Channel

A channel defined at an analysis server is related to a specific type of data to be captured at managed nodes, and provides a series of data to the analysis utilities. For example, a channel will receive CPU temperatures from the managed nodes at a regular interval. Another channel will receive error logs from the managed nodes. Analysis utility listens to one or more channels which provide the required data. Analysis utility can suspend receiving data from channels when the

analysis need not be performed. A channel is deactivated if it is not listened by any analysis utility. The analysis server notifies the managed nodes of the deactivation of a channel, and the managed nodes stop monitoring the data.

4.4. Execution Manager

The execution manager is the core of the runtime of the data analysis platform. It is responsible for executing analyzers. It schedules the analysis tasks based on their attributes as below.

1 Required time
2 Deadline
3 Importance

Required time of an analysis task is the estimated average execution time of the analyzer on a single server. Deadline of an analysis task is calculated by the execution manager based on the execution policy of the analysis. For example, execution policy of a fault predictor will be specified as “the analysis should be completed within 60 seconds since the data is captured.” Importance of an analysis task is calculated by the execution manager based on the type of the analyzer, the state of the node with which the task concerns, estimated reliability of the node, and elapsed time since the node was analyzed. An analysis task concerning the node which has been detached from the system because it was analyzed as unreliable will be assigned the lowest priority. An analysis task concerning the node which has been analyzed just before will be assigned a lower priority than the node which has not been analyzed for a longer time. An analysis task concerning the node whose estimated unreliability or anomaly has exceeded a threshold will be assigned a higher priority. The execution manager may discard low-prioritized analysis tasks to complete important analysis tasks in time.

Basically each analysis task is executed on a single node. In case an analyzer takes longer time by nature, the execution manager tries to execute such kind of analysis in parallel using multiple analysis servers if available. In case the analysis server becomes overloaded, the execution manager dispatches analysis tasks to other analysis servers.

4.5. Programming Model

Programming model of the data analysis platform is designed aiming to achieve scalable execution of analyzers without requiring deep knowledge of parallel programming to the developer of analyzer.

Figure 2 describes the data flow of an example analyzer. This analyzer calculates the difference of two input data from a statistical point of view. The analysis data contains a set of channel data which contains time series data collected just before from a resource for a specific interval. The reference data also contains a set of channel data. But they are extracted from a historical database. First, both the analysis data and the reference data are processed by the “resampling” module. The

“resampling” module takes a set of channel data and outputs the same number of channel data. Next, the output of the “resampling” module is splitted into individual channel data, and passed to the “preprocess” module. The “preprocess” module preprocesses a channel data for the following modules. The “Calc-A” module calculates a statistical value like average and mean value and outputs a scalar value. Finally, the “Calc-B” module combines each pair of scalar values and outputs them as an array of double. Each value of this array represents the difference of each channel data in the analysis data and the reference data. This array becomes the output of this analyzer.

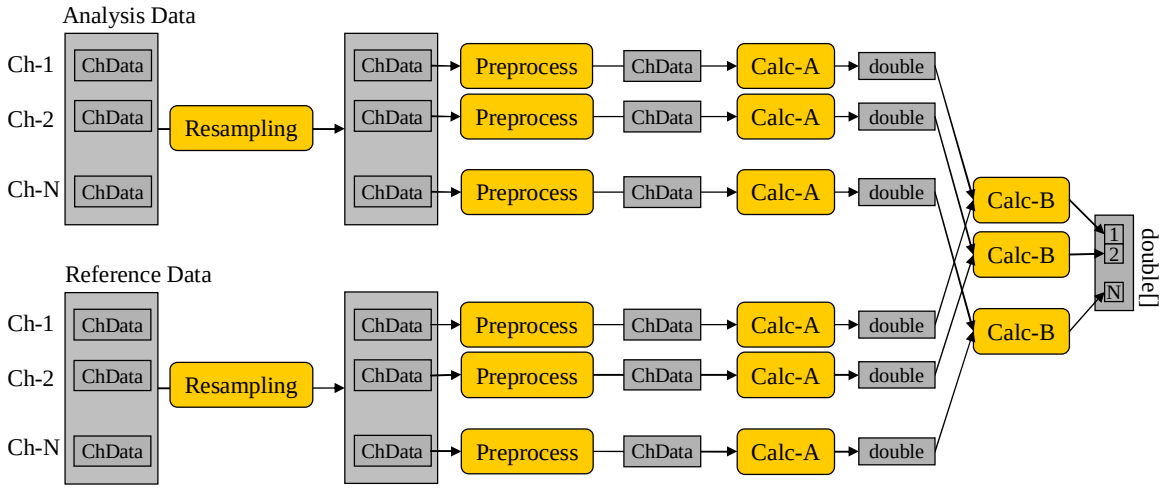

Figure 2: Data Flow of an Analyzer

Module is the smallest execution unit defined by the programming model. Users implement data transformations and data filters as a subclass of the module. The programming model provides some pre-defined modules which express parallel skeletons as below.

- Pipeline
- Parallel
- Map

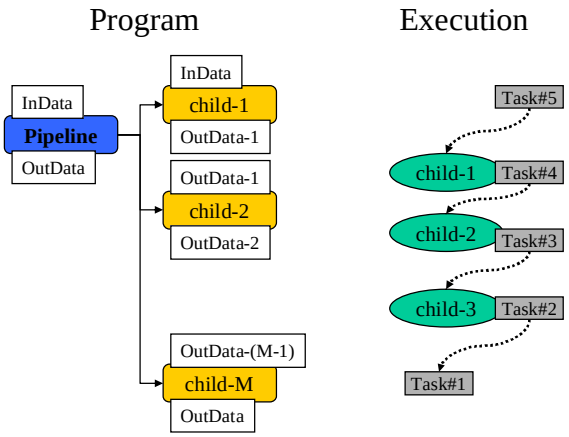

Figure 3: Pipeline Skeleton

Figure 3 describes the program notation and execution model of the “pipeline” skeleton. A “pipeline” skeleton contains sub modules as its children. It denotes that its children can be executed in parallel on a sequence of data stream. Output data of the last child becomes the output data of the parent pipeline.

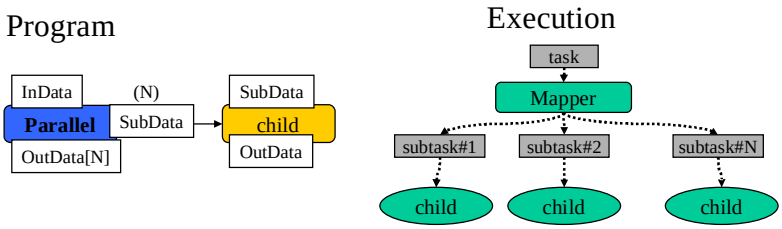

Figure 4: Parallel Skeleton

Figure 4 describes the program notation and execution model of the “parallel” skeleton. A “parallel” skeleton contains a sub module as its child. It denotes that input data is mapped to multiple sub-data, and they may be processed in parallel by the child module. A mapper defines how the input data is mapped to multiple sub-data. Users can use predefined mappers or they may define a mapper by themselves. Multiple data outputted by the child module are combined to an array, and it becomes the output data of the “parallel” skeleton.

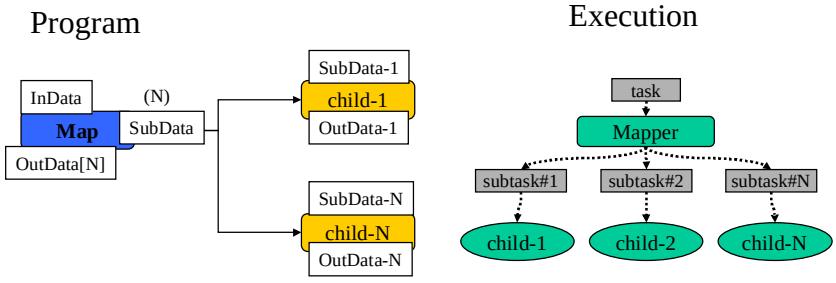

Figure 5: Map Skeleton

Figure 5 describes the program notation and execution model of the “map” skeleton. A “map” skeleton contains a sub module as its child. Its execution model is similar to the “parallel” skeleton. The “parallel” skeleton has only one child module and the splitted sub-data are processed by the same child module. On the other hand, the “map” module has multiple types of child modules so that the splitted sub-data can be processed differently by each child module. Children’s output data are combined to an array, and it becomes the output of the “map” skeleton.

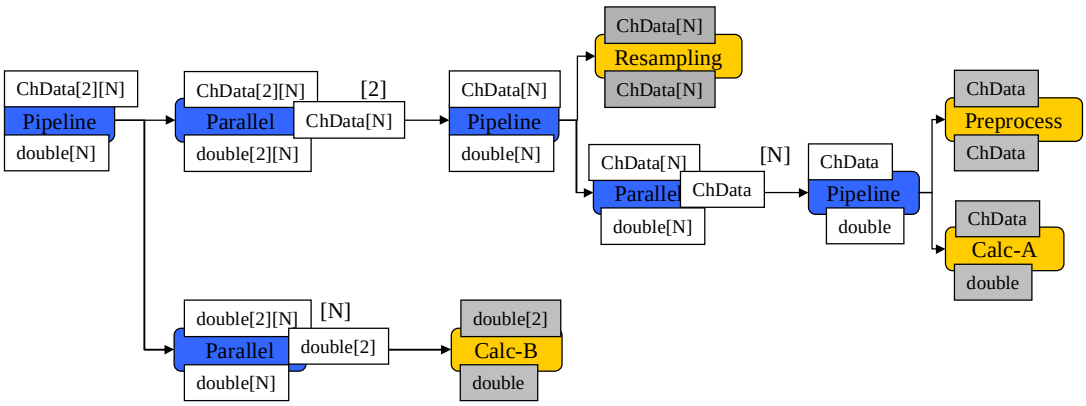

Figure 6: Program Notation of an Analyzer

Figure 6 describes the program of the example analyzer depicted in Figure 2 in a graphical notation. This program consists of 2 types of parallel skeletons (“pipeline” and “parallel”), and 4 types of user defined modules (“Resampling”, “Preprocess”, “Calc-A”, and “Calc-B”.) The “pipeline” skeleton at the top left is the root of this analysis program. This analyzer takes 2 sets of data each of which contains an array of channel data (“ChData[N]”,) and outputs an array of double. Each element of the array indicates the difference of each channel data. The first child of the top “pipeline” skeleton is a “parallel” skeleton. The “parallel” skeleton takes the same input data as the top “pipeline” skeleton and split the data into 2 sets of data which containing an array of channel data. The splitted data are passed to its child “pipeline” skeleton, and processed by descendant modules (“Resampling”, “Preprocess”, and “Calc-A”.) The first “parallel” skeleton under the top “pipeline” skeleton combines the data outputted by its child “pipeline” skeleton as a matrix of double. This array is passed to the sibling “parallel” skeleton. The sibling “parallel” skeleton transposes the array and splits into N sets of arrays of double. Each splitted data (array of double) is passed to child “Calc-B” module. The outputs of the “Calc-B” module are combined to an array, and it becomes the result of this analysis.

5. Summary

Failure management takes an important role for further advancement of productivity of large scaled systems. Failure prevention is one of the effective techniques to improve substantial performance and system utilization in the presence of failures. It takes proactive actions based on

fault predictions or anomaly detections. The data analysis platform provides a scalable infrastructure for the data analyzers for fault predictions and anomaly detections. The platform integrates scalable data collection mechanism and an analysis execution runtime aiming to achieve scalable execution of analyses using multiple servers in a soft real-time manner.

Problem determination becomes more difficult and intricate as the system scales more, thus administrator will require advanced tool that can indicate useful information from large number of historical data. The data analysis platform will also become an infrastructure for the data analyzers for problem determination. It enables to execute data analyses for problem determination on multiple servers so that analyses finish as fast as possible.

This study aims to establish a scalable data analysis platform for failure management, and contribute to further improvement of productivity of large scaled systems.

6. References

[1] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo, "Performance Implications of Failures in Large-Scale Cluster Scheduling," In Proceedings of 10th Workshop on Job Scheduling Strategies for Parallel Processing, June 2004.

[2] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. Parallel Computing, Vol. 30, Issue 7, July 2004.

[3] Matther J. Sottile and Ronald G. Minnich. Supermon: A high-speed cluster monitoring system. In Proceedings of International Conference on Cluster Computing, 2002.

[4] R. Bellofatto, P. G. Crumley, D. Darrington, B. Knudson, M. Megerian, J. E. Moreira, A. S. Ohmacht, J. Orbeck, D. Reed, and G. Stewart, “A database-centric approach to system management in the Blue Gene/L supercomputer,” In Proceedings of 20th International Parallel and Distributed Processing Symposium, 2006. pp.8-.

[5] J. M. Brandt, A. C. Gentile, D. J. Hale, and P. P. Pebay. OVIS: A Tool for Intelligent, Real-time Monitoring of Computational Clusters. In Proceedings of 20th International Parallel and Distributed Processing Symposium, 2006.

[6] Sahoo, R. K., Oliner, A. J., Rish, I., Gupta, M., Moreira, J. E., Ma, S., Vilalta, R., and Sivasubramaniam, A. 2003. Critical event prediction for proactive management in large-scale computer clusters. In Proceedings of the Ninth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining (Washington, D.C., August 24 - 27, 2003). KDD '03. ACM Press, New York, NY, pp. 426-435.

[7] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, Jose E. Moreira, and Manish Gupta. Filtering Failure Logs for a BlueGene/L Prototype. DSN, pp. 476-485, 2005.

[8] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, A. Sivasubramaniam: Fault-aware Job Scheduling for BlueGene/L Systems, In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), April 2004.

[9] Yawei Li, Zhiling Lan, "Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster Computing," pp. 531-538, Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006.

[10] Ann T. Tai, Kam S. Tso, William H. Sanders, Savio N. Chau: A Performability-Oriented Software Rejuvenation Framework for Distributed Applications. DSN 2005, pp. 570-579.

[11] Tsuyoshi Ide, Pairwise Symmetry Decomposition Method for Generalized Covariance Analysis, Proceedings of the fifth IEEE International Conference in Data Mining (ICDM 05), Nov 20-27, 2005, pp.657-660.