Differentiating Data Collection for Cloud Environment Monitoring

You Meng,Zhongzhi Luan, Zhendong Cheng, Depei Qian
Sino-German Joint Software Institute (JSI)
Beihang University (BUAA)
Beijing
you.meng@jsi.buaa.edu.cn

Abstract—**In a growing number of information processing applications, data takes the form of continuous data streams rather than traditional stored databases. Monitoring systems that seek to provide monitoring services in cloud environment must be prepared to deal gracefully with huge data collection without compromising system performance. In this paper, we show that by using a concept of urgent data, system can shorten the response time for most 'urgent' queries while guarantee lower bandwidth consumption. We argue that monitoring data can be treated differently. Some data capture critical system events, the arrival of these data will significantly influence the monitoring reaction speed, we call them urgent data. High speed urgent data collection would help system to act in real time when facing fatal error. On the other hand, slowing down the collection speed of others may render more bandwidth. Then several urgent data collection strategies that focus on reducing the urgent data volume are also proposed and evaluated to guarantee the efficiency.**

Keywords-cloud computing; cloud monitoring; urgent data; rule engine; constraint

I. INTRODUCTION

Monitoring services are essential to large-scale distributed systems due to its versatile applications. In cloud environments, various kinds of applications and facilities bring in even more objects to be monitored. One of the most important monitoring measurements is the response latency, which expresses the time between a stimulus and the response to it[10]. In cloud systems, the pursuit of high-quality services demands a response time as short as possible, because many systems can avoid the disaster by earlier alert. On the other hand, shorten the data collection interval, which is required by low latency of generating the alert, would directly generate higher pressure to the bandwidth consumption. This is especially unacceptable in production cloud system like EC2, because I/O is required by all the applications and measured as resources that directly generate revenues. The confliction between I/O occupations and low response latency seriously affects the performance of the monitoring system.

The responsibility of cloud monitoring systems can be classified into two parts. Firstly, the global status should be revealed by the monitoring system. The latency of obtaining these kinds of data can be less urgent because usually no more actions would be taken for these data. We call this kind of data as normal data. Another liability of the monitoring service is filtering the monitoring data streams and identifying the special data sequences that match the monitor rules, in other words, detecting anomaly. This kind of data always reveals that the system being monitored is in a bad state and should be interfered somehow. The data which demands the system to react as soon as possible are called urgent data. Since urgent data capture critical system events, the system response latency mainly depends on the quick arrival of the urgent data. Higher sampling frequency of the urgent data can help user to discover the fatal event as soon as possible. On the other hand, one or two minutes delay of the normal data would do nothing harm to the system but can render enough bandwidth to accelerate the urgent data collection.

In existing grid and cloud monitoring systems, monitoring information is typically sampled by local sensors and propagated to the repository, and then queried by users[5]. Other systems aim to provide real-time monitoring but limited by the rule execution or data quality[2,7,9] . Some other works[1] provides advanced query capabilities to monitoring system by using CEP as rule executor. Monitoring information is directly sent to the data center and verified by the rule engine. A distributed framework is also used for data reduction. But the response time of these systems is limited by the sampling speed. As the data volume of the system is huge, the sampling speed cannot be high due to bandwidth limitation. In contrast, using the urgent data scheme offers a more reasonable trade-off between the response latency and bandwidth consumption. Comparing to direct usage of the rule engine to process all data in each time step, SkyView uses a pre-executer before the rule engine to filter the whole dataset and pick-up a superset of the queried data. As the dataset volume is reduced, the data collection can be speeded up with the original bandwidth quota and the response latency of the monitoring system can be reduced.

The goal of this paper is to resolve the confliction between the response latency and the bandwidth consumption in the cloud environment. By changing the data collection speed of different data, we increase the reaction speed of the monitoring system while guaranteeing lower bandwidth consumption. A system, called SkyView which implements this idea, is introduced. SkyView is a HPC monitoring system that has monitored three largest supper-computer systems in China for years. SkyView makes contribution to the concept of urgent data monitoring in several ways.

978-3-901882-50-0

- Part of the monitoring data is recognized as the urgent data according to the user queries. The normal data is still sampled periodically but the collection speed is slowed down to reduce the bandwidth consumption.
- In order to solve the confliction between response latency and I/O, we focus on reducing the bandwidth consumption during the urgent data collection. Several strategies of executing the user queries are proposed to achieve this goal.

II. MOTIVATION AND SYSTEM ARCHITECTURE

Users always apply a batch of queries and demands to find the specific data set in the monitoring system. We use B as the original bandwidth consumption of the monitoring system, f as the data collection frequency and d_{all} as the data volume from each data collection process. So we get:

$$B = d_{all} \times f \quad . \tag{1}$$

Our solution is to change the data collection manner, the collected data d_{all} can be split into urgent data d_u and normal data d_n, respectively, the data volume in each data collection process is:

$$d_{all} = d_u + d_n \, . \tag{2}$$

Basically, all the monitoring data is sampled periodically by local sensors, and then used as both urgent data and normal data. However, the response latency is just dominated by the timely arrival of the urgent data. Instead of setting the same data collection frequency to all data, we collect more frequently the urgent data while slowing down the collection of the normal data. We use f_u and f_n to denote the collect frequency of urgent data and normal data, Eq.1 is modified into:

$$B = d_u \times f_u + d_n \times f_n \, . \tag{3}$$

This is the fundamental motivation of this paper. As the reaction latency of the monitoring system is mainly determined by the data collection interval of urgent data. And we aim to lower the system's response latency without increasing the bandwidth consumption, which is equivalent to increasing the frequency of urgent data collection f_u while keeping the bandwidth consumption B as a constant value. So there are only two ways to increasing f_u without changing B: decreasing the volume of urgent data d_u or decreasing the normal data collection frequency f_n. As slowing down the collection frequency f_n is easy to bring into effect, we mainly focus of reducing the data volume of urgent data.

Fig.1 is the architecture of the urgent data collection system. In the cloud environment, as the computing resources and data collection sensors are distributed placed, the data collection algorithms always consider them as N distributed nodes. First, the user queries are classified into normal queries and urgent queries according to query semantics. Second, the proper constraints is then selected from urgent queries and handled by data collection algorithms to collect the correspondent urgent data. At last urgent data and normal data are all sent to the rule executor in different frequency and the rule executor would generate the final query results.

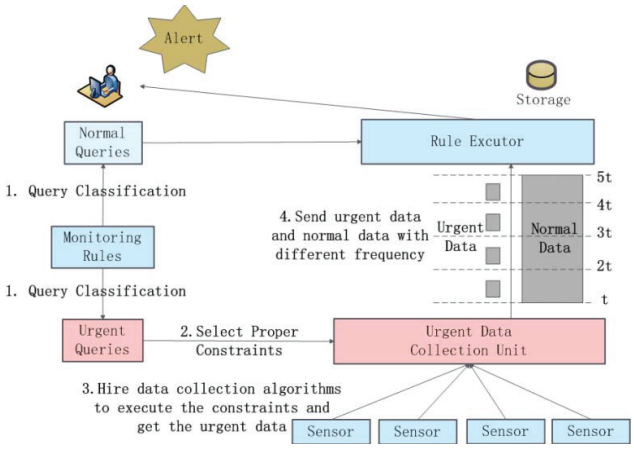

Figure 1 The SkyView Data Collection Architecture

III. URGENT DATA COLLECTION

In order to guarantee the high frequency of urgent data collection f_u, our goal is to minimize the communication cost d_u in each urgent data collection procedure. The urgent data collection procedure is supposed to just gather all the urgent data according to the urgent demand that extracted from the user queries, then the Data Collection Unit(DCU) send the urgent data and normal data in different frequencies. At last, all the data should be sent to the rule executor to generate the final result. As the cloud infrastructure is a so complicated, we consider it as a distributed environment, the problem can be present as followed:

Problem Definition: The user has specific constraints on monitoring data generated from N distributed node, and the system should return the corresponded data as results. Moreover, the monitoring system should guarantee the correctness and try best to lower the communication overhead.

A. The Extra Data Problem

In distributed environment, only collecting the urgent data without any extra bandwidth consumption is impossible. While collecting specific data, system should use some more data to judge whether a batch data belongs to the urgent data [3,4,11], this would also bring in bandwidth consumption, we call these transferred data as extra data. We use d_{real} to symbolize the real urgent data volume and d_{extra} to express the extra transferred data, so the bandwidth consumption in urgent data collection d_u should be changed to:

$$d_u = d_{extra} + d_{real} \, . \tag{4}$$

According to Eq.4, in order to reduce the bandwidth consumption d_u in urgent data collection, we have to minimize the volume of both the extra data and the real urgent data. The works that try to minimize the extra data according to single constraint has already been done and achieve highly efficiency.

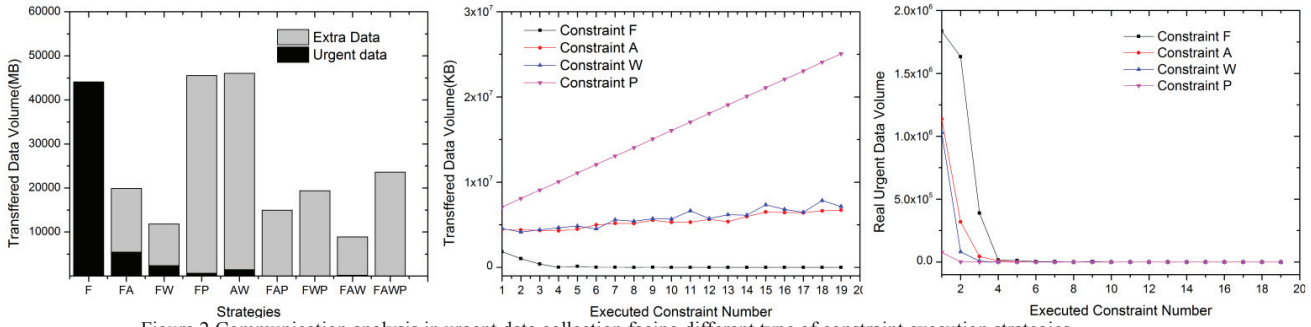

Figure 2 Communication analysis in urgent data collection facing different type of constraint execution strategies

B. *Multiple Constraints Execution Problem*

As the existence of extra data, we find that while an urgent data have several constraints on it, executing all the constraints may not get the lowest bandwidth consumption according to the experiments. In order to save more bandwidth in urgent data collection, we can consider only execute part of the constraints of the urgent data. According to this feature, we can judge whether a constraint should be send into DPU. This can be easily proofed by Eq.4, system want to minimize the communication in urgent data collection which is d_u, after adding a constraint in DPU, the system's extra data volume and the real urgent data volume would be changed. This constraint's extra data d_e should be added into the extra data, and the real urgent data should be decreased by the volume this constraint filtered d_f. The whole bandwidth consumption should be change to

$$d'_u = (d_{extra} + d_e) + (d_{real} - d_f) \tag{5}$$

If we want benefit from adding this constraint which is $d'_U < d_U$, according to Eq.4 and Eq.5 this constraint should satisfy $d_f > d_e$, which means this constraint must filter more data than its extra data. While facing multiple constraint execution problems, using the feature, system can easily find out which constraints should be executed in DPU and other constraints should be executed in rule executer.

Let's discuss this problem in a more general way. Supposing an urgent demand has k constraints, and the relation between the constraints is 'and'. If the raw data volume of this urgent data is D, let P_i be the data reduction percentage by executing constraint C_i locally, and E_i is the extra data volume of executing constraint C_i. The whole collected data volume DC_k should be:

$$DC_k = D\prod_{i=1}^{k}(1 - P_i) + \sum_{i=1}^{k} E_i \ . \tag{6}$$

Our purpose is to find the proper urgent demand extraction strategies to minimize transferred data volume DC_k. As P_i is determined by the data distribution and E_i is determined by algorithm used in executing the rule, we can only change k to minimize the total cost, which means we can decide how many constraints should be executed in DPU. If we have already extracted k constraints, and we want benefit from executing one more constraint C_{k+1}, which means $DC_{k+1} < DC_k$, bring it into Eq.6 we get:

$$E_{k+1} / P_{k+1} < D \times \prod_{i=1}^{k}(1 - P_i). \tag{7}$$

Noting the left part of Eq.7 is related to constraint C_{k+1}, others are only related to constraint C_1 to C_k. So this means the constraints with lower E_{k+1}/P_{k+1} would be easy to accept by the DPU.

C. *Constraint classification*

According to feature3, we have a specific standard to judge whether should let a constraint executed in DPU. But the extra data volume of a constraint is quite different when facing different data distribution and different constraints' threshold. System also can't determine the precise extra data volume until a constraint is executed. System has to find out an operational strategy to judge whether should execute a constraint in DPU. We further discover that the extra data volume is quite different while facing different type of constraints by experiments. So we classify the constraints into several types by the extra data volume percentage.

Fixed Property Constraint (F): The Constraint requires monitoring data from one node in one sampling procedure to satisfy corresponding requirements.

Aggregated Constraint (A): The Constraint requires data from different nodes in one sampling. The A-type constraints are most common constraints.

Window Constraint (W): Constraints that contain window functions which require the data from a specific time span satisfy some other constraints.

Patten Matching Constraint (P): Constraints that contain event pattern sequences.

After we classify the constraints into different types, system can directly judge whether should add a constraint into DPU by type. The extra data volumes of these constraints are further discussed in experiments.

IV. EXPERIMENT EVALUATION

We performed extensive experiments using multiple real world traces and also synthetic data to evaluate the performance of SkyView. We choose region-B of Magic Cube as our test bed. The region-B of Magic Cube consists of 650 nodes, 4376 Core, 200TB storage and computing power of 32Tflops. We highlight the most important observations in our experiments as follows

- Though the extra data volume is related to the data distribution and the algorithm that used. The different type of constraints also influences the extra volume a lot. So it is rational that we directly distinguish constraints by type.
- Constraint execution strategies significantly influence the performance of the urgent data collection and the reaction speed of the system.

Constraint Classification: We first compare the bandwidth consumption and the transferred data of different urgent constraints. We pick 3 for each type of them and marked as F(fix property filter), A(aggregated filter), W(window filter) and P(paten filter). The collected data satisfy all the constraints is expressed as "real urgent data", and the "extra data" symbolizes the other wasted data that algorithm collected. Fig.3 shows the total collected data with different constraints. According to different extra data percentages, constraints are grouped into three teams. Group1 only include the fix property filer F1~F3, because they do not generate any extra data. Group2 include the aggregated filter (A1~A3) and the window filter(W1~W3). As shown in Fig.3, these two types of constraints require the data collected from different node. So the extra data volume takes from 60% to 87% of the whole transferred data. But the whole volume of transferred data is still much smaller than the raw data, which means while using these constraints, system can reduce the monitored data volume. Group 3 only contains the pattern filters (P1~P3). As we choose "negative pattern" in the pattern filter, system always need every data to judge whether the answer is correct. The extra data in pattern filter takes nearly 100% of the whole data volume.

Multi-constraints urgent data execution strategies: As shown in the former experiment, the performances of the constraints are quite the same in type. So we just use constraint type instead of the specific constraint, and evaluate the combination of them in different format. The combination strategies are expressed by name, for example, executing both fix property filter (F) and aggregated filer(A) would get strategy "FA", and extracting all the four type constraints would get "FAWP". Fig.2 shows the communication cost and communication breakup with of different constraint combination. As shown in figure, the collected data volume is the same in "FA", "FW" and "FAW" which consume less bandwidth than others. The strategy "FWP" and "FWAP" would get less real urgent data. But as shown in the collected data volume, these two strategies consume the most data due to too much extra data, because the execution efficiency of pattern filter (P) is too low as we verified in the former experiment. Use strategy "F" would get the correspondent data in 100%, but the whole volume is still too large. Though the former three strategies ("FA", "FW" and "FAW") cost nearly the same transferred data. The useful data percent is much different. As window filter containing the time window, it usually needs more transferred data to judge whether a constraint is satisfied. On the other hand, aggregated filter just need instantaneous data from different nodes, so the combination "FA" get higher useful data percent than "FW". The combination of "FAW" would filter much more data, but

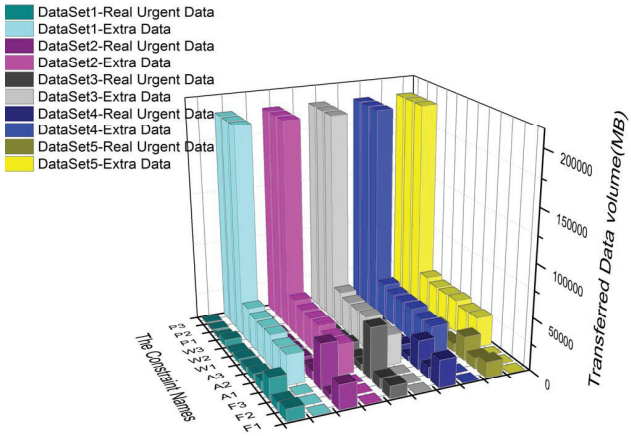

Figure 3 The Constraint Execution Performance Analysis with different Dataset and different

as the constraints is more the former two, the extra data also takes too much transferred data.

V. CONCLUSION AND FUTURE WORK

This paper presents a new solution to shorten the response latency of monitoring by using urgent data. We showed that monitored data can be divided into urgent data and normal data, accelerating the collection speed of urgent data can directly shorten the response latency of the whole monitoring system. We also pointed out that the constraint execution strategies are key issue that influences the efficiency of the system, executing every constraint would not gain the best efficiency. Several strategies are proposed and evaluated. Though we divided the constraints into four types, many other query semantics like top-k or skyline are not discussed here. As part of our ongoing work, we are trying to evaluate them in the future work.

REFERENCES

[1] B. Balis, B. Kowalewski, M. Bubak, "Real-time Grid monitoring based on complex event processing" in FGCS 27(8), 1103–1112 (2011)

[2] A. Benharref, M. A. Serhani, and S. Bouktif "Online Monitoring for Sustainable Communities of Web Services", in IM 2011

[3] M. Wu, J. Xu, X. Tang, and W.-C. Lee, "Monitoring top-k query in wireless sensor networks," in ICDE, 2006.

[4] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi, "Efficient detection of distributed constraint violations," in ICDE, 2007.

[5] S. Andreozzi, N.D. Bortoli, and M.C. Vistoli," GridICE: a monitoring service for Grid systems", in FGCS 21 (4) (2005) 559–571.

[6] M. Smith, F. Schwarzer, and B. Freisleben, "A Streaming Intrusion Detection System for Grid Computing Environments" In HPCC, 2009

[7] D. Jurca, and R. Stadler "Computing Histograms of Local Variables for Real-Time Monitoring using Aggregation Trees" in IM,2009

[8] H. Wright, R. Crompton, S. Kharche, P. Wenisch: Steering and visualization, "Enabling technologies for computational science" in FGCS 26(3), 506–513 (2010)

[9] F. Fusco, F. Huici, L. Deri, and T. Ewald "Enabling High-Speed and Extensible Real-Time Communications Monitoring" in IM, 2009

[10] http://www.memidex.com/latency+reaction-time

[11] S. R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla, "Efficient constraint monitoring using adaptive thresholds," in ICDE, 2008.