Azerbaijan Journal of High Performance Computing, Vol 1, Issue 2, 2018, pp. 133-139
https://doi.org/10.32010/26166127.2018.1.2.133.139

Scalable Complex Event Processing Using Rule Distribution

Mohsen Sharifi, Mohammad Ali Fardbastani

Distributed Systems Research Lab, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, msharifi@iust.ac.ir, fardbastani@iust.ac.ir

*Correspondence: Mohsen Sharifi, Distributed Systems Research Lab, School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, msharifi@iust.ac.ir

Abstract

Complex event processing (CEP) systems are currently widely used in large-scale enterprises for the processing of high and dynamically changing rates of input events using large number of complex rules. Given the hardware limitations of vertically scaled CEP solutions, horizontal scalability has become an essential requirement for modern CEP systems. In this paper, we propose an adaptive load-balancing technique via rule distribution (called ARD) for a cluster of CEP engines that provides horizontal scalability for CEP systems. Our experiments show our proposed technique provides higher scalability and yields higher throughput in comparison with two previously proposed non-adaptive load-balancing techniques, namely VISIRI and SCTXPF, when the system faces with variable workload. In addition, ARD keeps the system balanced more often.

Keywords: complex event processing, scalability, CEP, horizontal scaling, load balancing, throughput.

1. Introduction

In event-driven systems, real-time detection of complex patterns from a large amount of producing events and derivation of higher level events is an essential requirement. This type of information processing is called complex event processing (CEP) [1]. Processing rules describe complex event patterns in CEP.

CEP is employed in many domains such as in big data analysis [2] [3], business process management [4], smart cities and internet of things [5], [6]. In such domains, a CEP system must process high rates of input events to detect high number of complex patterns using the priori specified processing rules. To heighten the performance of the CEP system, one can distribute the tasks of a CEP system among multiple compute nodes.

Some of the previously proposed techniques decompose rules to basic operators and distribute processing of the operators among compute nodes. These solutions have many limitations in rule decomposition and do not support all

types of CEP rules. Furthermore, these solutions have their own new rule syntaxes that stops users to use their priori known rule syntaxes of popular CEP engines. Another category of techniques distribute CEP rules among compute nodes, each of which runs an instance of a CEP engine. In these techniques, users can employ their favorite CEP engine and use a priory-known rule syntax. But the previously proposed techniques distribute rules statically at the startup of the CEP systems and are not adaptable with the changes in the system workload.

In this paper, we propose an adaptive rule distribution (ARD) technique for scalable complex event processing. ARD distributes CEP rules among clustered compute nodes and continuously monitors resource usages of nodes to keep their loads balanced when the system workload changes.

We have organized the rest of the paper as follows. Section 2 presents our adaptive load balancing technique. Section 3 presents the results of our experiments. Section 4 presents related works and Section 5 concludes the paper.

2. Adaptive Rule Distribution (ARD)

In this section, we explain how ARD technique distributes the rule set of a CEP system among a clustered set of compute nodes and adaptively keeps the load of the system balanced. First, we present the architecture of ARD and then demonstrate how system load is calculated in ARD. Finally, we present our load balancing technique.

2.1. Architecture

Figure 1 shows the architecture of ARD. The CEP system consists of a coordinator and many CEP nodes. The coordinator distributes rules among the CEP nodes and continually monitors resource utilization of the CEP nodes. When resource utilizations of CEP nodes changes due to changes in their workloads, the load of system becomes imbalanced and coordinator migrates some rules from high-load nodes to low-load nodes till the system gets rebalanced again.

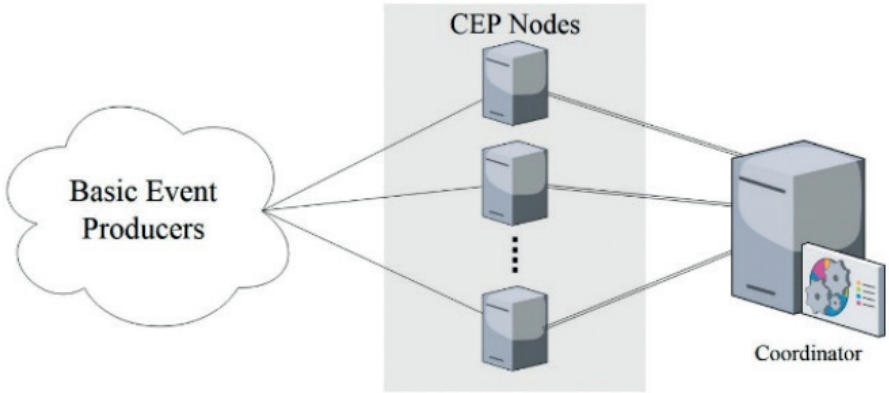

Figure 1. ARD Architecture

2.2. Load Calculation

For making a correct decision for load balancing, the current value of resource utilization should not be used. Because making a decision based on the instant value is not the best decision. Therefore, we use the exponential moving average (EMA) to take part the history of resource utilizations in the calculation of system load and load balancing. Equation 1 shows the calculated load of resource r at each node.

$$L_r^i = (1 - \alpha) * L_r^i + \alpha * C_r^i \quad (1)$$

In Equation 1, L_r^i stands for the calculated load of resource r of node i and C_r^i stands for the current value of utilization of resource r of node i.

We use standard deviation to calculate the system imbalance. If the system has n nodes and each node has m types of resources, we calculate the imbalance value (I) of the system as given by Equation 2.

$$I = {}^1/_m \Sigma_{r=1}^m \sqrt{{}^1/_n \Sigma_{i=1}^n (L_r^i - {}^1/_n \Sigma_{i=1}^n L_r^i)^2} \quad (2)$$

2.3 Load Balancing

For load balancing in ARD, the system continually monitors the load of the nodes. When changes in the system workload imbalances the load of the system, rules are migrated between the nodes until the system becomes rebalanced. Balance status of the system is checked periodically. In each period, the load balancing procedure (Figure 2) is executed. In each period, the load balancer checks the imbalance value of the system. If the value is more than a predefined threshold, the system is considered to be in an imbalance state. Therefore, a rule is randomly selected from the node with the most average resource utilization and moved to the node with the least average resource utilization. In the next period, the imbalance value is checked again and if the system is still in an imbalance state, another rule is moved. This rule migration is repeated until the system becomes balanced again.

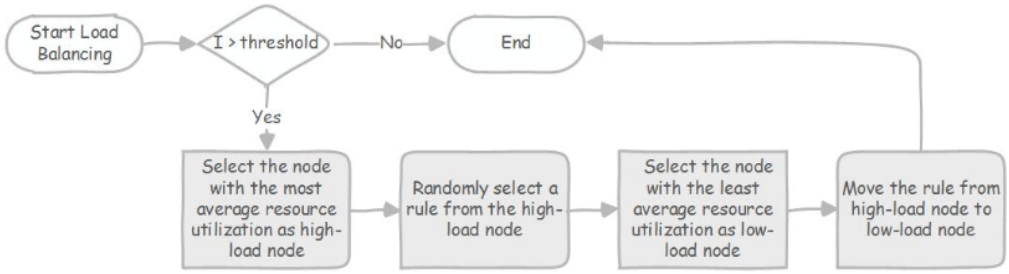

Figure 2. ARD Load Balancing Flow

3. Evaluation

We have implemented ARD using Java language and evaluated its scalability

and adaptability via some test scenarios. Our testbed consists of one coordinator node and a number of CEP nodes, each of which is a virtual machine that runs over VMware ESXi 6. The operating system of all nodes is Ubuntu server 16.04 and the Java version is 1.8. Coordinator has 4 processing cores and its Java heap has 2 GB of memory. Each CEP node has 2 processing cores with 2GB of memory for its Java heap. Drools Fusion [7] is deployed on CEP nodes as the CEP engine of our evaluation. Drools Fusion is a popular Java-based open-source CEP engine developed by the Red Hat Inc.

The event set of our tests consists of 32 event types. The rule set of the evaluation consists of 1000 rules and is a random subset of all permutations of the event types, and conjunction (&), disjunction (|), negation (~) and sequence operators. In addition, all rules have a time-window.

To show the adaptability of ARD, the generation rate of each event type is random and total event generation rate is constant in each experiment. Furthermore, the generation rate of each event type varies every 10 minutes, and the duration of the experiments are 60 minutes.

The results of evaluation of ARD is compared with two other static load-balancing techniques. The first is SCTXPF[8] that balances the number of rules on CEP nodes. The second is VISIRI[9] that uses a rule cost estimation to distribute rules among CEP nodes according to the estimated costs.

To evaluate scalability, we evaluate the throughput with respect to the number of CEP nodes. We test a single CEP node several times with different workloads to find the maximum throughput of a single CEP node. Then we repeat the experiment with more number of nodes and the total event rate is increased with respect to the

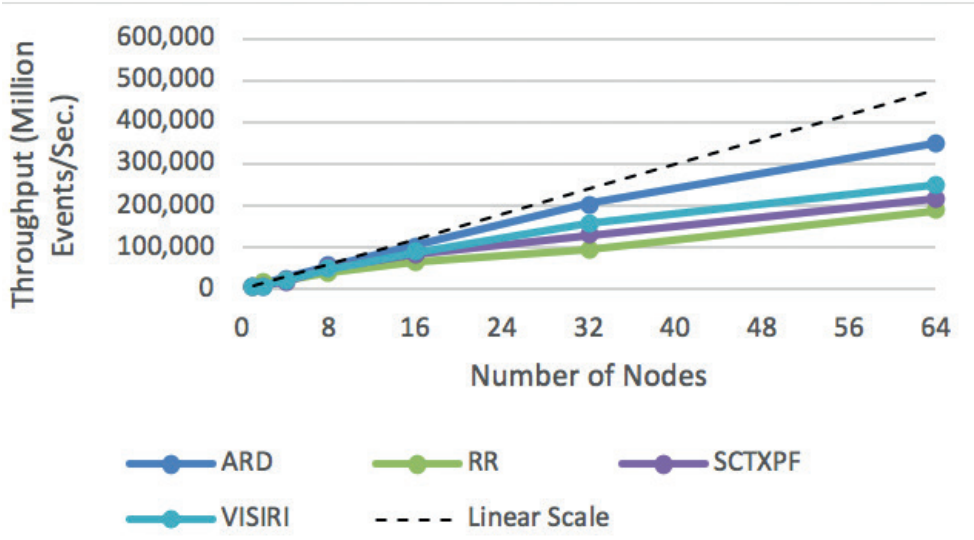

Figure 3. Throughput evaluation

number of CEP nodes. Figure 3 shows the increase in the throughput of the system when the number of nodes is increased linearly. As Figure 3 shows, in a variable workload, the throughput of ARD increases higher than in the other techniques when the number of CEP nodes increases.

To compare the load balancing of ARD, SCTXPF, and VISIRI, we calculate the average imbalance of the system for every 10 minutes. As Figure 4 shows, in addition to yielding lower system imbalance, ARD adaptively handles changes to system workloads. In contrast, changes in the workload in each period cause notable variation in the system imbalance when using SCTXPF and VISIRI techniques.

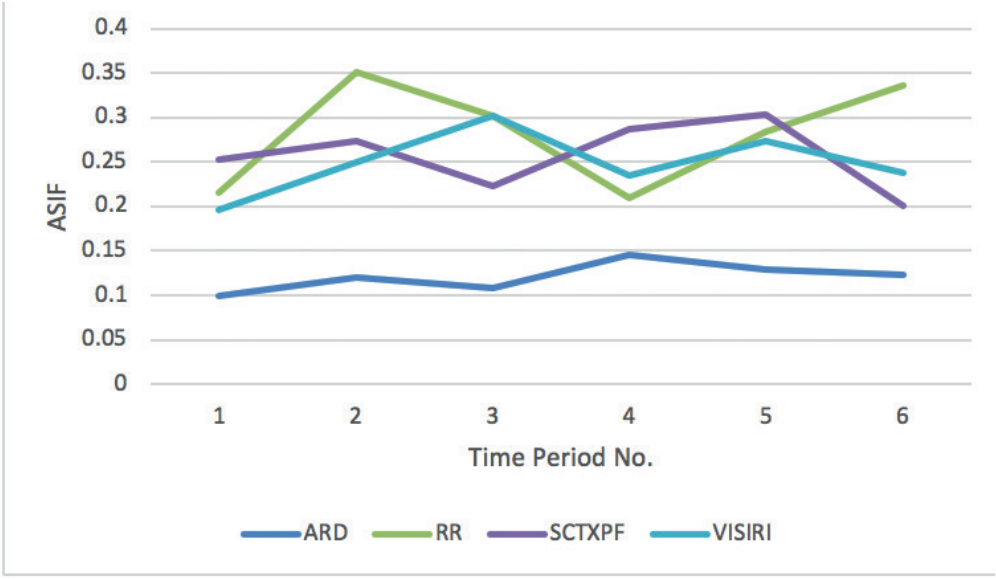

Figure 4. Load balance evaluation

4. Related Work

Several techniques such as the work reported in [10] have been proposed for parallelization of CEP on a single machine with multiple multi-core CPUs. However, because of hardware limitations of vertical scaling of complex event processing, horizontal scaling of CEP is an essential requirement to counter a high rate of input events and a large number of complex rules.

The previously proposed techniques for horizontal scaling of complex event processing can be categorized in two categories. In the first category of horizontal scaling techniques such as those reported in [11]–[17], rules are decomposed to their basic CEP operators and operators are distributed among many compute nodes. The main shortcomings of these techniques are decomposing limitation of all types of CEO rules, load balancing and overloading of an operator and high communication overhead between compute nodes.

The second category is clustering of CEP engines. In this category of horizontal

scaling of CEP, users can use popular CEP engines and their familiar rule syntaxes. Some of the previously proposed techniques for CEP clustering such as those reported in [8], [9], [18], [19] only balance the load of the system statically via distributing the rules among the compute nodes. Therefore they do not handle changes in the system workload pattern during the lifetime of the system.

In [4], an adaptive load balancing for some types of CEP rules in business process monitoring is proposed. However, this is not a general adaptive load balancing technique.

5. Conclusion

Given to the rise in the rates of input events and the number of rules in CEP systems, scalability of CEP systems is an essential requirement. In this paper, we propose an adaptive load balancing technique, called RAD, via rule distribution among a scalable cluster of CEP engines. In ARD, resource utilization of all compute nodes is monitored and if the system goes to an imbalance state because of changes in the workload, rules are migrated to rebalance the system. Our experiments shows higher system throughput under ARD, with increases in the number of compute nodes, in comparison with two other non-adaptive load-balancing mechanisms. In addition, our experiments show that ARD makes the system more balanced when the workload varies during system operation. As future work, one can develop a fully distributed CEP system by removing the singular coordinator, which can become a single point of failure, and distributing its tasks among the computing nodes that run the CEP engines.

References

[1] Dayarathna, M., & Perera, S. (2018). Recent Advancements in Event Processing. *ACM Computing Surveys, 51*(2), 1-36. doi:10.1145/3170432

[2] Zhang, P., Shi, X., & Khan, S. U. (2018). Quantcloud: Enabling big data complex event processing for quantitative finance through a data-driven execution. *IEEE Transactions on Big Data.*

[3] Shi, S., Jin, D., & Tiong-Thye, G. (2017). Real-time public mood tracking of Chinese microblog streams with complex event processing. *IEEE Access, 5,* 421-431.

[4] Fardbastani, M. A., Allahdadi, F., & Sharifi, M. (2018). Business process monitoring via decentralized complex event processing. *Enterprise Information Systems, 12*(10), 1257-1284.

[5] Bonino, D., & De Russis, L. (2018). Complex Event Processing for City Officers: A Filter and Pipe Visual Approach. *IEEE Internet of Things Journal, 5*(2), 775-783.

[6] Graubner, P., Thelen, C., Körber, M., Sterz, A., Salvaneschi, G., Mezini, M., Seegar, B., Freisleben, B. (2018, June). Multimodal Complex Event Processing on Mobile Devices. In *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems* (pp. 112-123). ACM.

[7] Browne, P. (2009). *JBoss drools business rules.* Birmingham: Packt.

[8] Isoyama, K., Kobayashi, Y., Sato, T., Kida, K., Yoshida, M., & Tagato, H. (2012, July). A scalable complex event processing system and evaluations of its performance. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems* (pp. 123-126). ACM.

[9] Kumarasinghe, M., Tharanga, G., Weerasinghe, L., Wickramarathna, U., & Ranathunga, S. (2015, June). VISIRI-Distributed Complex Event Processing System for Handling Large Number of Queries. In *International Conference on Coordination Languages and Models* (pp. 230-245). Springer, Cham.

[10] Fathollahzadeh, S., Teymourian, K., & Sharifi, M. (2016, June). Stateful complex event detection on event streams using parallelization of event stream aggregations and detection tasks. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems* (pp. 390-393). ACM.

[11] Mayer, R., Slo, A., Tariq, M. A., Rothermel, K., Gräber, M., & Ramachandran, U. (2017, December). Spectre: Supporting consumption policies in window-based parallel complex event processing. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (pp. 161-173). ACM.

[12] Mayer, R., Tariq, M. A., & Rothermel, K. (2017, June). Minimizing communication overhead in window-based parallel complex event processing. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems* (pp. 54-65). ACM.

[13] Gong, Y., Kuang, H., Cai, X., Hu, H., Song, W., & Lu, J. (2017, June). Parallelized Mobility-Aware Complex Event Processing. In *2017 IEEE International Conference on Web Services (ICWS)* (pp. 898-901). IEEE.

[14] Dwarakanath, R. C., Koldehofe, B., & Steinmetz, R. (2016, December). Operator Migration for Distributed Complex Event Processing in Device-to-Device Based Networks. In *M4IoT Middleware* (pp. 13-18).

[15] Fonseca, J., Ferraz, C., & Gama, K. (2016, June). A policy-based coordination architecture for distributed complex event processing in the internet of things: doctoral symposium. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems* (pp. 418-421). ACM.

[16] Weisenburger, P., Luthra, M., Koldehofe, B., & Salvaneschi, G. (2017, May). Quality-aware runtime adaptation in complex event processing. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on* (pp. 140-151). IEEE.

[17] Wang, Q., & Shang, Y. (2019). A Distributed Complex Event Processing System Based on Publish/Subscribe. In *Recent Developments in Intelligent Computing, Communication and Devices* (pp. 981-990). Springer, Singapore.

[18] Kobayashi, Y., Isoyama, K., Kida, K., & Tagato, H. (2015, June). A complex event processing for large-scale M2M services and its performance evaluations. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems* (pp. 336-339). ACM.

[19] Pathak, R., & Vaidehi, V. (2015, May). An efficient rule balancing for scalable complex event processing. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 190-195).

Submitted 25.06.2018
Accepted 12.10.2018