Auto-Scaling Techniques for Spark Streaming

Master-Thesis von Seyedmajid Azimi Gehraz
Tag der Einreichung:

1. Gutachten: Prof. Dr. rer. nat. Carsten Binnig
2. Gutachten: Dr. Thomas Heinze

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Data Management

Auto-Scaling Techniques for Spark Streaming

Vorgelegte Master-Thesis von Seyedmajid Azimi Gehraz

1. Gutachten: Prof. Dr. rer. nat. Carsten Binnig
2. Gutachten: Dr. Thomas Heinze

Tag der Einreichung:

Contents

1 Abstract

4 Reinforcement Learning

5 Design

7 Evaluation

Dynamic resource scaling in cloud environments has been studied extensively in literature. As a naive implementation, there are two thresholds, namely *upper bound* and *lower bound*. However, such an implementation suffers from *oscillating* decisions. In order to remedy this issue, *grace period* shall be enforced. During this period, no scaling decision is made.

Hasan et al. [8] introduced four thresholds and two time periods. `ThrUpper` defines upper bound. `ThrBelowUpper` is slightly below `ThrUpper`. Similarly, `ThrLower` defines lower bound and `ThrAboveLower` is slightly above the lower bound. In case, system utilization stays between `ThrUpper` and `ThrBelowUpper` for a specific duration, then cluster controller decides to take a scale-out action, by adding resources. On the other hand, if system utilization stays between `ThrLower` and `ThrAboveLower` for a specified duration, then controller decides to take scale-in action. Defining two levels of thresholds helps to detect workload *persistency* and avoid making immature scaling decision. However, defining thresholds is a tricky and manual process, and need to be carefully done[4]. It should be noted that, computation overhead of this approach is very low.

RightScale applies voting algorithm[20] among nodes to make scaling decision. In order for a specific action to be decided, majority of nodes should vote in favour of that action. Otherwise, no action is selected as a default action. Afterwards, nodes apply grace period to stabilize the cluster. The complexity of the voting process in trusted environments is in the order of $O(n^2)$, which leads to heavy network traffic among participants when cluster size grows. This approach is also categorized in threshold based approaches. Thus, it suffers from the same issue as mentioned above.

Herbst et al. [11] surveys different auto-scaling techniques based on time-series analysis in order to forecast *trends* and *seasons*. *Moving Average Method* takes the average over a sliding window and smooths out minor noise level. Its computational overhead is proportional to size of the window. *Simple Exponential Smoothing*(SES) goes further than just taking average. It gives more weight to more recent values in sliding window by an exponential factor. Although it is more computationally intensive compared to moving average, it is still negligible. SES is capable of detecting short-term trends but fails at predicting seasons. These approaches are more specific instances of *ARIMA* (Auto-Regressive Integrated Moving Average) which is a general purpose framework to calculate moving averages. However, time-series analysis is only suitable for stationary problems consist of recurring workload patterns such as web applications. In case of streaming (specially in multi-tenant environments), in which the workload is highly unpredictable, time-series analysis tends to be less useful. Additionally, more advanced forms of time-series analysis which are capable of forecasting seasons (such as *tBATS Innovation State Space Modelling Framework*[15], *ARIMA Stochastic Process Modelling Framework*[13]) are computationally infeasible for streaming workloads.

Herbst et al. [10] continues the survey on state of the art techniques to predict future workload. It includes workload forecasting based on *Bayesian Networks*(BN) and *Neural Networks*. There are several issues with each of them that makes them unsuitable for streaming workloads. As an example, there is no universally applicable method to construct a BN. Furthermore, it requires collecting data and training the model offline. Neural networks suffer from the same issues. That is, it requires collecting samples and training the model offline. For complex models, training phase is typically computationally infeasible which is conflicting with requirements of thesis.

Tesauro et al. [23] proposes a hybrid approach to overcome poor performance of online training. The system consists of two components: an online component based on queuing system combined with reinforcement learning component that is trained offline. The offline component is based on *neural networks*. Author models the data center as multiple applications managed under a single resource manager. Modelling streaming workloads as a queuing system has two problems. First, modelling is a complicated process and determining probability distributions requires domain knowledge. Second, it requires access to each node (so it can be modelled as a queue) which is currently not possible without modifying `spark-core` package. Since, it was one the requirements to provide a solution without making any modification to `spark-core`, this work has been abandoned.

Rao et al. [19] proposed to use reinforcement learning to manage resources consumed by virtual machines. It employs standard model-free learning, which is known as *temporal difference*[22] or *sarsa* algorithm. The state space consists of metrics collected from virtual machines (CPU, RAM, Network IO, . . .). There is no global controller and each node decides based on its own Q-Table. As mentioned in literature, standard temporal difference has a slow convergence speed. In order to speedup bootstrap phase, Q-Table is initialized by values that were obtained during separate supervised training. Since this approach also relies on offline training, it wasn't adopted by this thesis.

Heinze et al. [9] implemented reinforcement learning in the context of FUGU[7] (which is a *data stream processing* framework). Each node, has its own Q-Table and imposes local policy without coordinating to other nodes. This architecture can not be applied in context of spark streaming, since spark asbtracts away individual nodes from perspective of applicaiton developer. In order to decrease state space, the author applies two techniques. First, only system utilization is considered. Second, system utilization is discretized using coarse grained steps. To remedy slow convergence, controller enforces *monotonicity constraint*[12]. That is, if controller decides to take scale-out action for a specific utilization, it may not decide scale-in for even worse system utilization. This feature has been adopted by this thesis.

Enda, Enda, and Jim [6] proposed a parallel architecture to reinforcement learning. Standard model-free learning (temporal difference) is used. No global controller is involved and each node decides locally. In order to speed up learning, all nodes, maintain two Q-Tables (local and global tables). Local table is learned and updated by each node. Whenever, an agent learns a new value for a specific state, it broadcasts it to other agents. Global table contains values received from other agents. Additionally, agent prioritize local and global tables by assigning weights to each table. Weights are factors that are defined by application developers. Final decision is the outcome of combining local and global tables. Although each node learns some part of the state space (which may overlap with other nodes), it is not applicable in the context of spark streaming. The assumption in this architecture is that, each node is operating autonomously without intervention of other nodes (such as web servers). In contrast, spark is a centrally managed system. That is, all nodes running spark jobs are supervised by a single master node (probably with couple of backup masters).

Cardellini et al. [2] proposed a two level hierarchical architecture for resource management in Apache storm[21]. There is a local controller on each node which is cooperating with a global controller. Local controller monitors each operator using different policies (threshold-based or reinforcement learning using temporal difference). In case, local controller decides to scale in or out an specific operator, it contacts the global controller and informs it about its decision. Then it waits to receive confirmation from global controller. Global controller operates using a token-bucket-based policy[3] and has global view of cluster. It ranks requests coming from local controllers and either confirms or rejects their decisions. Although. this architecture seems to be a promising approach, however it has been implemented by modifying Storm's internal components. As mentioned above, this is in conflict with thesis's requirements.

In order to mitigate the problem of large state space in reinforcement learning, Lolos et al. [18] proposed to start the agent from small number of coarse grained states. As more metrics are collected (and stored as historical records), agent will discover *outlier* parameters (those parameters that are affecting agent more, CPU rather than IO as an example). Then, it partitions the affected state into two states and *re-trains* newly added states using historical records. Both temporal difference and value iteration methods can be used as learning algorithm. Gradually, agent only focuses on some specific parts of the state space, since all parameters are not equally important. This approach, effectively reduces the size of state space. However, the trade-off is the storage cost in which historical metrics need to be stored. It worth noting that from the context of paper, storage cost (whether it is in-memory or on-disk and the duration of storing historical metrics) is unclear. Thus, this approach has been abandoned due to uncertainty.

Dutreilh et al. [5] proposed a model-based reinforcement learning approach for resource management of cloud applications. All virtual machines are supervised by a single global controller. Slow convergence is bottleneck of model-free learning, in contrast to model-based learning. However, environment dynamics are not available at time of modelling. Authors proposed to estimate these parameters as more metrics are collected and then switch to *value iteration*[22] algorithm instead of *temporal difference*. In short, for each visit of (state, action, reward, state) quadruple as (s, a, r, s'), the following state variables will be stored:

$$CountStateAction(s,a) = CountStateAction(s,a) + 1$$
$$RewardStateAction(s,a) = RewardStateAction(s,a) + r$$
$$CountStateTransition(s,a,s)] = CountStateTransition(s,a,s') + 1$$

Periodically two variables, namely $\overline{T}$ and $\overline{R}$, are computed as follows:

$$\overline{T(s,a,s')} = \frac{CountStateTransition(s,a,s')}{CountStateAction(s,a)}$$
$$\overline{R[s,a]} = \frac{RewardStateAction(s,a)}{CountStateAction(s,a)}$$

As more metrics are collected, $\overline{T}$ and $\overline{R}$ become more accurate and can be directly used in *Bellman* equation. Until enough measurements get collected, a separate initial reward function is used which is essentially the original reward function but with penalty costs removed. Furthermore, In order to reduce the state space – tuple of [request/sec, number of VMs, average response time) – there exists a predefined upper and lower bound for state variables and average response time is measured at granularity of seconds. This approach has been partially adopted by this thesis.

Lohrmann, Janacik, and Kao [16] proposed a solution based on queueing theory. The solution is designed for *Nephele*[17] streaming engine which has a master-worker style architecture. Similar to spark-streaming, a job is modelled as a DAG. It utilizes *adaptive output batching*[24] – which is essentially a buffer with variable size – to buffer outgoing messages emited from one stage to the other. Each task – an executer that runs UDF – is modelled as a G/G/1 queue. That is, the probability distributions of message inter-arrival and service times are unknown. In order to approximate these distributions, a formula proposed by Kingman [14] is used. From a bird's eye view, this solution seems promising. However, authors made two unconvincing assumptions that led us to abandon the proposal. First, worker nodes shall be homogeneous in terms of processing power and network bandwidth. Second, there should be an effective partitioning strategy in place in order to load balance outgoing messages between stages. In reality both assumptions rarely occur. Large scale stream processing clusters are build incrementally. Depending on workload, data skew does exist and imperfect hash functions are widely used by software developers.

Dutreilh et al. [4] proposed a model-free reinforcement learning approach (*Temporal difference* algorithm) with modified *exploration* policy. The standard exploration policy for Q-Learning is $1-\epsilon$. Under this policy, the agent performs a random action with probably of ϵ and with probability of $1-\epsilon$, it adheres to an action proposed by optimal policy. Although the random action is necessary to explore unknown states, but it has sever consequences under streaming workloads. In some cases, it leads to unsafe states where SLOs are severely violated. Since streaming is heavily latency sensitive, this property is undesirable. Thus, author sought toward a heuristic-based policy proposed by Bodik et al. [1]. This policy is based on couple of key observations.

- It must quickly explore different states.
- It should collect accurate data as fast as possible, to speedup training.
- During exploration phase, the policy should be careful not to violate SLOs.

The aforementioned policy, works as follows.

- Initially, policy pushes the system towards its maximum capacity by removing as many worker VMs as possible, until some SLO is violated.
- As soon as SLO is violated, it *immediately* adds one VM to compensate SLO violation.
- It keeps a pool of hot-standby worker VMs, to help recover from undesirable decisions.
- Adding and removing VMs one at a time has a major benefit. It helps to approximate maximum throughput of a single worker VM. This approximation assists the policy to figure out, how much SLO would be impacted in case it adds or removes one virtual machine.
- During grace period, measurements are discarded. The logic behind such a behaviour is that, during stabilization period metrics tend to be less accurate.

Some of the key concepts of this work, has been adopted by the author of this thesis.

[1] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson. “Automatic Exploration of Datacenter Performance Regimes”. In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*. ACDC ’09. Barcelona, Spain: ACM, 2009, pp. 1–6.

[2] V. Cardellini, F. L. Presti, M. Nardelli, and G. R. Russo. “Decentralized self-adaptation for elastic Data Stream Processing”. In: *Future Generation Computer Systems* 87 (2018), pp. 171 –185.

[3] V. Cardellini, F. Lo Presti, M. Nardelli, and G. Russo Russo. “Towards Hierarchical Autonomous Control for Elastic Data Stream Processing in the Fog”. In: *Euro-Par 2017: Parallel Processing Workshops*. Ed. by D. B. Heras, L. Bougé, G. Mencagli, E. Jeannot, R. Sakellariou, R. M. Badia, J. G. Barbosa, L. Ricci, S. L. Scott, S. Lankes, and J. Weidendorfer. Cham: Springer International Publishing, 2018, pp. 106–117.

[4] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck. “From Data Center Resource Allocation to Control Theory and Back”. In: *2010 IEEE 3rd International Conference on Cloud Computing*. 2010, pp. 410–417.

[5] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck. “Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow”. In: *7th International Conference on Autonomic and Autonomous Systems (ICAS’2011)*. Venice, Italy, May 2011, pp. 67–74. URL: https://hal-univ-paris8.archives-ouvertes.fr/hal-01122123.

[6] B. Enda, H. Enda, and D. Jim. “Applying reinforcement learning towards automating resource allocation and application scalability in the cloud”. In: *Concurrency and Computation: Practice and Experience* 25.12 (2012), pp. 1656–1674.

[7] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. “Multi-resource Packing for Cluster Schedulers”. In: *SIGCOMM Comput. Commun. Rev.* 44.4 (2014), pp. 455–466.

[8] M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, and S. L. D. Gudreddi. “Integrated and autonomic cloud resource scaling”. In: *2012 IEEE Network Operations and Management Symposium* (2012), pp. 1327–1334.

[9] T. Heinze, V. Pappalardo, Z. Jerzak, and C. Fetzer. “Auto-scaling techniques for elastic data stream processing”. In: *2014 IEEE 30th International Conference on Data Engineering Workshops*. 2014, pp. 296–302.

[10] N. Herbst, A. Amin, A. Andrzejak, L. Grunske, S. Kounev, O. J. Mengshoel, and P. Sundararajan. “Online Workload Forecasting”. In: *Self-Aware Computing Systems*. Ed. by S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu. Cham: Springer International Publishing, 2017, pp. 529–553.

[11] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn. “Self-adaptive Workload Classification and Forecasting for Proactive Resource Provisioning”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ICPE ’13. Prague, Czech Republic: ACM, 2013, pp. 187–198.

[12] H. Herodotou and S. Babu. “Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs”. In: *Proceedings of the VLDB Endowment*. Vol. 4. Jan. 2011, pp. 1111–1122.

[13] R. Hyndman and Y. Khandakar. “Automatic Time Series Forecasting: The forecast Package for R”. In: *Journal of Statistical Software, Articles* 27.3 (2008), pp. 1–22.

[14] J. F. C. Kingman. “The Single Server Queue in Heavy Traffic”. In: *Proceedings of the Cambridge Philosophical Society* 57 (1961), p. 902.

[15] A. M. D. Livera, R. J. Hyndman, and R. D. Snyder. “Forecasting Time Series With Complex Seasonal Patterns Using Exponential Smoothing”. In: *Journal of the American Statistical Association* 106.496 (2011), pp. 1513–1527.

[16] B. Lohrmann, P. Janacik, and O. Kao. “Elastic Stream Processing with Latency Guarantees”. In: *2015 IEEE 35th International Conference on Distributed Computing Systems*. 2015, pp. 399–410.

[17] B. Lohrmann, D. Warneke, and O. Kao. “Nephele streaming: stream processing under QoS constraints at scale”. In: *Cluster Computing* 17.1 (2014), pp. 61–78.

[18] K. Lolos, I. Konstantinou, V. Kantere, and N. Koziris. “Elastic Resource Management with Adaptive State Space Partitioning of Markov Decision Processes”. In: (2017).

[19] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. “VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration”. In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC ’09. Barcelona, Spain: ACM, 2009, pp. 137–146.

[20] RightScale. *Set up Autoscaling using Voting Tags*. Accessed June 20, 2018. 2018. URL: `http://support.rightscale.com/12-Guides/Dashboard_Users_Guide/Manage/Arrays/Actions/Set_up_Autoscaling_using_Voting_Tags/`.

[21] A. Storm. *Apache Storm*. Accessed June 21, 2018. 2018. URL: `http://storm.apache.org/`.

[22] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. The MIT Press, 1998. ISBN: 0262193981.

[23] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. “A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation”. In: *2006 IEEE International Conference on Autonomic Computing*. 2006, pp. 65–73.

[24] D. Warneke and O. Kao. “Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud”. In: *IEEE Transactions on Parallel and Distributed Systems* 22.6 (2011), pp. 985–997.