Tejo: A Supervised Anomaly Detection Scheme for NewSQL Databases

Guthemberg Silvestre, Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun

Tejo: a supervised anomaly detection scheme for NewSQL databases

Guthemberg Silvestre[1,2], Carla Sauvanaud[1,3], Mohamed Kaâniche[1,2], and Karama Kanoun[1,2]

[1] CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[3] Univ de Toulouse, INSA de Toulouse, LAAS F-31400 Toulouse, France
{gdasilva,csauvana,mohamed.kaaniche,karama.kanoun}@laas.fr

Abstract. The increasing availability of streams of data and the need of auto-tuning applications have made big data mainstream. NewSQL databases have become increasingly important to ensure fast data processing for the emerging stream processing platforms. While many architectural improvements have been made on NewSQL databases to handle fast data processing, anomalous events on the underlying, complex cloud environments may undermine their performance. In this paper, we present Tejo, a supervised anomaly detection scheme for NewSQL databases. Unlike general-purpose anomaly detection for the cloud, Tejo characterizes anomalies in NewSQL database clusters based on Service Level Objective (SLO) metrics. Our experiments with VoltDB, a prominent NewSQL database, shed some light on the impact of anomalies on these databases and highlight the key design choices to enhance anomaly detection.

1 Introduction

Big data has transformed the way we manage information. As an unprecedented volume of data has become available, there is an increasing demand for stream processing platforms to transform raw data into meaningful knowledge. These velocity-oriented platforms may rely on cloud databases to provide fast data management of continuous and contiguous flows of data with horizontal scalability. Therefore, cloud databases represent an important technology component for a broad range of data-driven domains, including social media, online advertisement, financial trading, security services, and policy-making process.

The architecture of row-store-based relational databases has evolved to meet the requirements of big data on the cloud [19], like elasticity, data partitioning, shared nothing, and especially high performance. The so-called NewSQL databases offer high-speed, scalable data processing in main-memory with consistency guarantees through ACID (atomicity, consistency, isolation, and durability) transactions.

To ensure fast data management, NewSQL databases rely on built-in, fault-tolerance mechanisms, like data partitioning, replication, redundant network topologies, load balancing, and failover. Although these mechanisms handle fail-stop failures successfully, many other cloud performance anomalies may remain unnoticed [20]. For instance, Do *et al.* [8] found that a single limping network interface can cause a three orders of magnitude execution slowdown in cloud

databases. Therefore, we believe that the dependability of NewSQL databases might be improved by detecting these anomalies.
This paper proposes Tejo, a supervised anomaly detection scheme for NewSQL databases. We make three specific contributions. First, we introduce a scheme for analysing performance anomalies using fault injection tools and a supervised learning model. Second, we shed some light on the impact of performance anomalies in NewSQL databases. Third, we highlight the importance of selecting the proper features and statistical learning algorithm to enhance the anomaly detection efficiency on these databases.
In the next section, we lay out the recent trends in data stream processing and anomaly detection with statistical learning. Following this, in Section 3 we describe the design of Tejo, in particular its components and its two-phased functioning, namely learning and detection phase. In Section 4 we evaluate VoltDB, a prominent NewSQL database, using Tejo. In our experimental setup, VoltDB served two workloads, whose data was partitioned and replicated across a cluster of virtual machines (VMs). Finally, we discuss the related work in Section 5, and conclude in Section 6.

2 Background

2.1 Big data stream processing and NewSQL databases

To processing continuous streams of big data, we consider the emerging stream processing platforms [15], as depicted in Figure 1. In these platforms, streams of data are processed by two complementary systems: the fast stream processing system and big archival engine. The former manages high-speed data streams to provide real-time analytics and data-driven decisioning, providing services like fraud heuristics, market segmentation, or optimal customer experience; while the later computes huge volumes of historical data for long-term data analytics, such as scientific results, seasonal predictions, and capacity planning. Big archival engines are built on data warehouse technologies like Hadoop and column-stores. In contrast, fast stream processing technologies are still emerging. Among these technologies are NewSQL databases like VoltDB [23] and S-Store [4]. To support incremental, stateful ingest of data streams into a scalable system, NewSQL databases provide low-latency via in-memory distributed processing and a strong support for transaction management with ACID guarantees. However, as NewSQL databases are deployed on cloud infrastructures to scale to large clusters, cloud performance anomalies may undermine their capacity of fast stream processing.

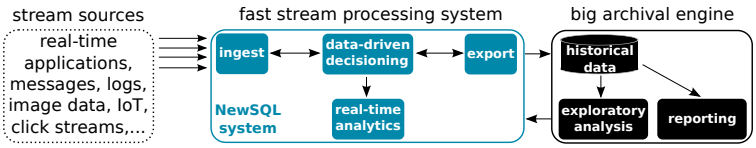


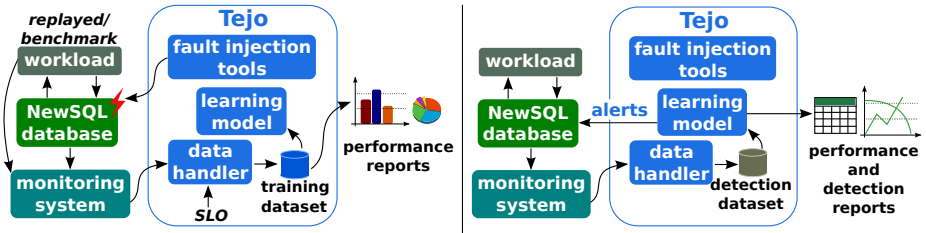
Fig. 1: The emerging stream processing platforms for big data.

2.2 Anomaly detection using statistical learning

Statistical learning has been a widely used technique to predict performance anomalies in large-scale distributed systems [5]. It makes prediction by processing feature vector $\mathbf{x}$ with a fixed number of dimensions d ($\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$) from the input space $\mathcal{X}$. There are two main methods: *supervised* and *unsupervised learning*.
The *supervised learning method* couples each input with a y, a label, from the output space $\mathcal{Y}$. To learn, we have N pairs $(\mathbf{x}, y)$ drawn *independent and identically distributed* (i.i.d.) from a fixed but unknown joint probability density function $Pr(X, Y)$. This method searches for a function $f : \mathcal{X} \to \mathbb{R}$ in a fixed function class $\mathcal{F}$ in the learning dataset. State-of-the-art algorithms, like *random forests* [2], aim to find $f^\star$ in $\mathcal{F}$ with the lowest empirical risk $f^\star \in \arg\min_{f \in \mathcal{F}} \mathtt{r}_{emp}(f)$, where $\mathtt{r}_{emp}(f) = \frac{1}{N} \sum_{i=1}^{N} I_{\{f(\mathbf{x}) \neq y_i\}}$ is computed over the training set, and $I_{\{.\}}$ is the indicator function which returns 1 if the predicate $\{.\}$ is true and 0 otherwise. Similarly, an *unsupervised learning method* relies on N unlabelled samples having probability density function $Pr(X)$. Unlike supervised learning, predictions provide insights into how the data is organized or clustered.
Most of the anomaly detection approaches for distributed systems are based on a general-purpose, unsupervised learning method [13,14,12]. However, prediction efficiency remains the main drawback of this method [16]. Results in our previous work [21] confirm that a supervised learning method overcomes an unsupervised one in cloud anomaly detection. In this work, we extend our supervised learning model as a component of Tejo to classify anomalous VMs in four different classes.

3 Approach

Tejo comprises three components, namely a set of fault injection tools, a data handler, and a learning model. These components interoperate into two distinct phases: learning and detection phase. While the first phase allows us to evaluate the performance of a NewSQL database under anomalies, the second permits the detection of these anomalies. Figure 2 depicts the components and the two-phase functioning of Tejo.



(a) Learning phase. (b) Detection phase.

Fig. 2: Tejo operates in two distinct phases: learning and detection phase.

3.1 The components of Tejo

Fault injection tools. To provoke performance anomalies, this component emulates four categories of faulty events in VMs of a NewSQL database cluster.

Network faults. Communication issues are common in distributed systems. To analyse their impact, we inject three types of network faults, namely packet loss, network latency, and limping network. Packet loss and network latency emulates interconnection issues, such as network partition. Limping network reproduces anomalies previously observed by Do *et al.* [8], where the transmission rate of limping network interface is smaller than the manufacturer's specification.
Memory faults. As NewSQL databases fit the entire data to the main memory, they become more vulnerable to anomalies in memory availability. To provoke such anomalies, we make arbitrary amounts of main memory unavailable. As a result, the database instance is likely to perform more costly disk I/O operations. A typical example of this fault is a VM running out of memory due to a misconfiguration, memory leaking, overloading, or an unbalanced resource allocation.
Disk faults. Although most NewSQL databases manage data in main memory, disk-intensive processes may have an impact of its performance. This category of fault emulates an arbitrary number of jobs performing several disk operations, including writes, reads, and file syncs.
CPU faults. CPU is a key resource in a virtual machine. As unattended number of processes compete the database instance to CPU resources, they may undermine the performance of the database cluster. The CPU fault emulates an arbitrary number of jobs performing arithmetic operations to overload the VM cores.

The data handler. This component computes data from the monitoring system (i) to collect the performance counters of a NewSQL database and (ii) to provide data to characterize performance anomalies.
Collect the performance counters of a NewSQL database. The data handler samples monitoring data to collect the current state of the NewSQL cluster, as depicted in Figure 2. To this end, it frequently communicates with the monitoring system to fetch raw monitoring data and to convert it into useful, aggregated information. The content of the resulting aggregated information depends on the aim of each functioning phase, learning or detection, detailed below.
Providing data to characterize performance anomalies. After aggregating samples of monitoring data, the data handler organizes this data into *feature vectors*. These vectors represent the state of the VMs of the database cluster or the workload. The vectors are stored in datasets for performance analysis or anomaly detection.

The learning model. The learning model is at the heart of the Tejo scheme. The purpose of this model, the so-called predictive task, is to characterize the behaviour of VMs under performance anomalies. Given an i.i.d. sample $(\mathbf{x}, y)$, described in Subsection 2, we model our predictive task as a classification problem, whose inputs and outputs are defined as follows.
Inputs. We represent the input space $\mathbf{x}$ as a VM running a database instance. This input data corresponds to a feature vector computed by the data handler component. The size of the feature vector matters. In general, the higher the dimension of this vector, the higher the predictive efficiency is. However, an increase in the input dimension rises the computational cost of predictions.
Outputs. The supervision y associated to each input VM $\mathbf{x}$ is based on five possible classes, $\mathcal{Y} \in \{0, 1, 2, 3, 4\}$, whose labels are *normal*, *network-related anomaly*,

memory-related anomaly, *disk-related anomaly*, and *CPU-related anomaly* respectively. Depending on the phase of Tejo (detailed below), these labels are assigned by either computing the training dataset or by a learning algorithm.

3.2 Two-phase functioning

Learning phase. In this phase, Tejo learns the behaviour of the database cluster under anomalies and reports on its performance.
Requirements. As illustrated in Figure 2a, Tejo relies on an already existing monitoring system to poll performance counters from both VMs and workload. To measure the cluster-wide performance counters, we assume that the workload can be replayed or run through a benchmark tool. We consider that Tejo's analyst specifies expected SLO metrics, such as average throughput and 99^{th} percentile latency. The analyst must also specify parameters of the fault campaign and running experiments, including intensity and duration of each fault, number of injections, and interval between consecutive fault injections.
Functioning. As the replayed/benchmark workload runs, the fault injection tool performs a fault injection campaign to emulate performance anomalies. Meanwhile, performance counters from both the workload and VMs running the database cluster are collected by the monitoring system and computed by the data handler component. As the data handler computes the monitoring data in feature vectors, it adds information about injected faults and labels. Labels correspond to output classes of learning model and are added with respect to SLO metrics. The feature vectors are then stored in the training dataset. After running the workload and accomplishing the fault injection campaign, the learning model computes the feature vectors of VMs from the training dataset.
Reports. Besides providing data to learn the behaviour of anomalous VMs, analysts can observe the impact of anomalies in the throughput and 99^{th} percentile latency. They may evaluate which anomalies cause SLO violations, gaining more insight into the efficiency of existing fault-tolerance mechanisms.

Detection phase. In this phase, Tejo reports on the efficiency of the learning model and performs anomaly detection in VMs at runtime.
Requirements. Similar to the training phase, Tejo relies on an existing monitoring platform to gather data for predictions. It requires that the learning model has already been trained as detailed in learning phase described above. We assume that SLO targets and the workload are the same as those of the learning phase.
Functioning. While the NewSQL database serves the workload, the data handler gathers the monitoring data and creates feature vectors of VMs in the detection dataset. As soon as a new feature vector is created, the learning model computes it to detect performance anomalies whenever they occur.
Reports. Tejo's learning model predicts labels of incoming feature vectors. Then alerts are generated about detected anomalies. These alerts may be handled by the database to trigger recovery procedures. Besides generating alerts, it reports on the efficiency of the learning model, including comparing different learning algorithms, ranking performance counters with regard to their importance, calculating the computational cost, and verifying model over-fitting or under-fitting.

4 Evaluation

We evaluate VoltDB, a NewSQL database, with Tejo. First, we describe our experimental setup. Second, we measure the impact of performance anomalies in a VoltDB cluster. Finally, we report on the predictive efficiency of these anomalies.

4.1 Experimental setup

We performed our experiments on a private cloud consisting of two Dell PowerEdge R620 hosts. Each host has two-core Xeon E5-2660 at 2.2 GHz, 64 GB of memory, and two 130 GB SATA disks. Hosts are connected by Gigabit Ethernet. We chose VMware as the virtualization technology and ESXi 5.1.0 as hypervisor. Figure 3 depicts our private cloud, highlighting the consolidation of VMs. Each VM of the NewSQL database cluster has 4 GB of memory, 4 CPU cores, a disk of 16 GB, and is connected to a 100Mbps virtual network.

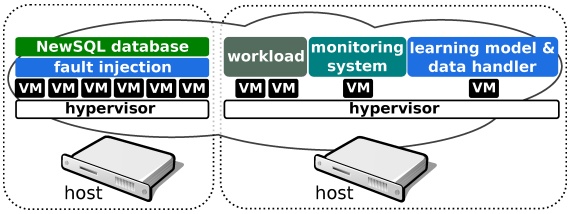


Fig. 3: Experimental setup.

Components of Tejo and monitoring system. As fault injection tools, we chose Dummynet (v3.0) [3] for network faults and stress-ng (v0.01.30) [4] for disk, memory, and CPU faults. These tools provide a flexible, easy-to-reproduce way to inject arbitrary fault intensities. Table 1 lists the parameters of our fault injection campaign. The data handler component was implemented as a collection of python/shell scripts along with PostgreSQL database for datasets. We implemented our learning model using the Scikit-learn library [17], from which we evaluated three learning algorithms: random forests [2], gradient boosting [10], and SVM [7]. We used Ganglia as monitoring system. Our setup required additional Ganglia plug-ins for collecting performance counters of the workload and VoltDB. Every 15 seconds, we collected 147 performance metrics of each VM, and the average throughput and the 99^{th} percentile latency from the served workload.

NewSQL database and workloads. We evaluated VoltDB (v4.x) as NewSQL database. We set the number of partitions VoltDB to 18 across a cluster of six VMs with failover mechanisms enabled. We varied the replication degree k from two to zero (i.e., replication disabled). We evaluated VoltDB with two workloads, the popular TPC-C benchmark for OLTP [5], and Voter [6], a workload derived from leaderboard maintenance application for Japanese version of the "American Idol".

[4] stress-ng. `http://kernel.ubuntu.com/~cking/stress-ng/`
[5] TPC-C benchmark (v5.10). `http://www.tpc.org/tpcc/`
[6] Voter. `https://github.com/VoltDB/voltdb/tree/master/examples/voter.`

Table 1: The key parameters of Tejo for our fault injection campaign.

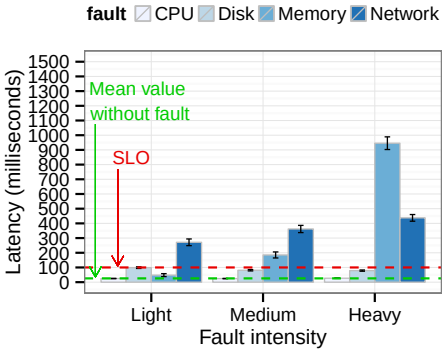
Fault		**Intensity ranges**			**Unit**
		Light	Medium	Heavy	
Network	Pkt loss	1.6-3.2	4-5.6	6.4-8	%
	Latency	8-20	26-38	44-56	ms.
	Limping	85-65	56-38	29-11	Mbps
Memory		73-79	82-88	91-97	%
Disk		10-20	25-35	40-50	writers
CPU		19-39	49-69	79-99	%

4.2 Evaluating performance anomalies in VoltDB

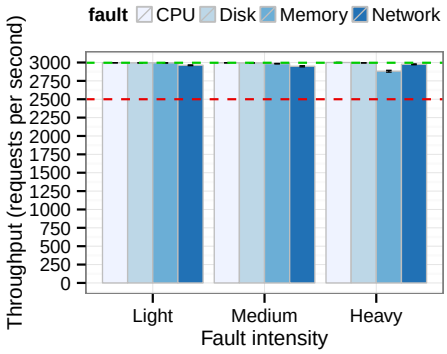
We run Tejo in learning mode(Subsection 3.2) to evaluate the impact of faults in VoltDB. We selected a dataset containing 200,000 samples, including performance counters of VMs and the workload. Data was evenly collected across the two evaluated workloads. Figure 4 shows the impact of faults on the performance of VoltDB with a replication degree k=2. For each workload, they show the resulting performance anomalies on the average throughput and 99^{th} percentile latency, including mean values without faults, the expected SLO metrics, and 95% confidence interval for performance metrics under fault injection.

Overall, the impact of increasing levels of faults was higher on the 99^{th} percentile latency than the average throughput. For instance, Figure 4a shows that the 99^{th} percentile latency of VoltDB serving Voter workload soars under faults, especially for network and memory faults. Although the mean of the 99^{th} percentile latency without fault was 25 milliseconds, it reaches 945 milliseconds under memory faults. Similar results were found as VoltDB served TPC-C workload. However, we noticed that TPC-C has a greater performance degradation under memory faults (Figure 4c). The reason for that is the main memory usage of each workload. While Voter uses 25% of main memory from each VM, TPC-C utilises almost 50%. Consequently, TPC-C is more sensitive to memory faults than Voter. Disk faults had a limited impact of the performance of VoltDB, slightly higher on TPC-C than Voter due to a greater need to synchronize data from the main memory to disk (Figure 4c). Surprisingly, CPU faults had no impact on the performance of both workloads, even under heavy fault intensity (i.e., 99% of CPU usage).

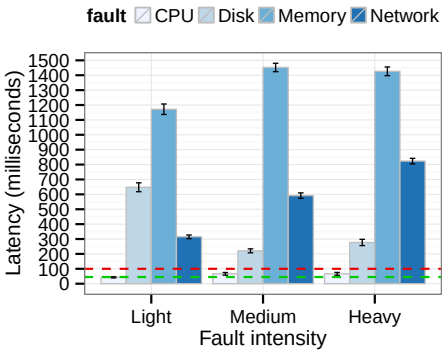
To shed some light on the capacity of data replication to mitigate the impact of performance anomalies, we varied the replication degree k of VoltDB from two to zero (i.e., replication disabled). Figure 5 shows a summary of the results of the impact of faults with medium intensity on VoltDB. In general, our results suggest that higher the replication degree the worse is the performance. The reason is that NewSQL databases as VoltDB strive to provide ACID properties both for concurrent transactions and for replicas. The impact of a fault on a single node spreads across the replicas on the cluster more easily, worsening its performance.



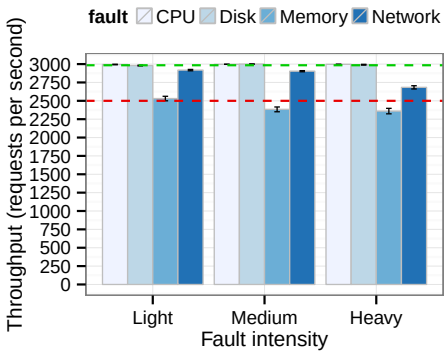
(a) 99^{th} percentile latency w/ Voter.



(b) Average throughput w/ Voter.

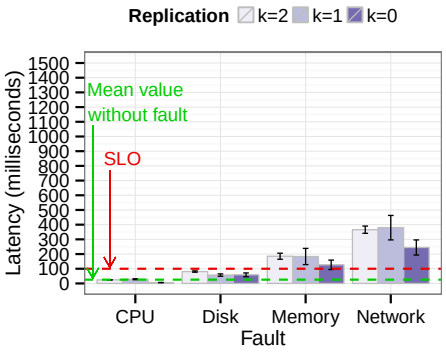


(c) 99^{th} percentile latency w/ TPC-C.

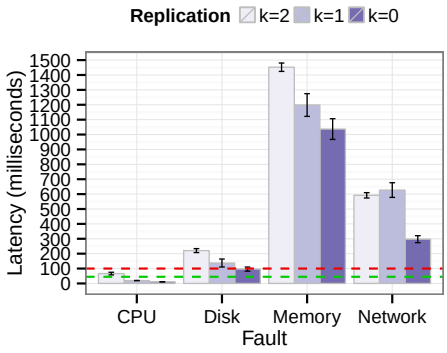


(d) Average throughput w/ TPC-C.

Fig. 4: Performance anomalies in VoltDB as serving Voter and TPC-C, with a replication degree k=2.



(a) Voter workload.



(b) TPC-C workload.

Fig. 5: Medium fault impact on VoltDB with different k values.

4.3 Predictive efficiency analysis

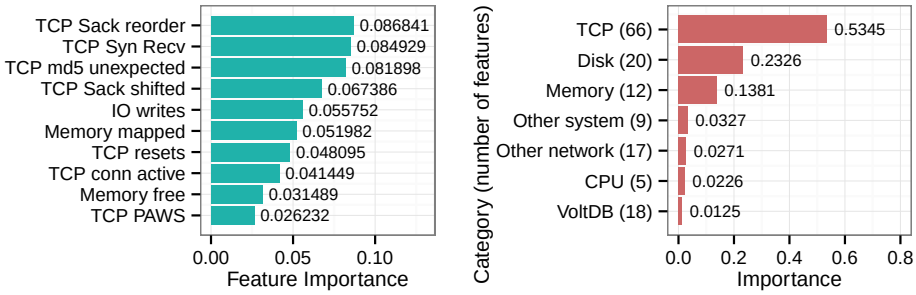
We evaluated the predictive efficiency of learning model of Tejo with three learning algorithms, random forests, gradient boosting, and SVM. To this end, we used data derived from the dataset of Subsection 4.2. The derived data was computed by the Tejo's data handler, as described in Subsection 3.2. Each new sample had 147 features ($d = 147$, as discussed in Subsection 2.2) and a label corresponding to a class of Tejo's learning model (see Subsection 3.1). Recall that anomaly-related labels are only assigned to samples that violated the SLO. The resulting dataset contained 10,000 samples for each evaluated workload, including 5,000 of samples representing anomalous events in VMs. To validate the learning model properly, this dataset was split in two uneven parts: three-fifths of data for training the model and two-fifths for testing its predictive efficiency. We used two well-known measures to evaluate the learning model efficiency, precision and F1-score. We also computed the overhead of predictions with each learning algorithm.

Table 2 summarizes our results. Regardless the learning algorithm, the learning model of Tejo was able to detect 96% of anomalies properly. It performed better with random forests algorithm, whose overall score was 0.99 (up to 1) for both precision and F1-score measures. Random forests also provided the lowest overhead for anomaly detection, requiring less than 30 microseconds for a prediction. The SVM algorithm had the worst predictive performance, particularly to detect memory-related anomalies. SVM also incurred the highest overhead for anomaly detection with our model, performing two orders of magnitude slower. According to Friedman [10], this happens because SVM shares the disadvantages of ordinary kernel methods, such as poor computational scalability and inability to deal with irrelevant features. In contrast, boosting methods, like random forests and gradient boosting, overcome these issues by using a linear combination of (many) trees.

Table 2: Anomaly detection performance with different learning algorithms.

Algorithm	**Class**	**workload**					
		voter			**TPC-C**		
		precision	F1-score	overhead	precision	F1-score	overhead
Random Forests	Normal	0.99	0.99		0.98	0.99	
	Network	0.98	0.98	23μs	0.99	0.98	26μs
	Memory	0.99	0.98		0.98	0.99	
	Disk	0.99	0.98		1.00	1.00	
Gradient Boosting	Normal	0.99	0.99		0.96	0.98	
	Network	0.99	0.99	30μs	0.99	0.96	33μs
	Memory	0.99	0.99		0.98	0.99	
	Disk	1.00	1.00		1.00	1.00	
SVM	Normal	0.98	0.97		0.98	0.98	
	Network	0.97	0.96	4294μs	0.99	0.97	5441μs
	Memory	0.85	0.91		0.87	0.93	
	Disk	1.00	0.97		1.00	1.00	

In addition to the predictive efficiency evaluation, Tejo allows us to analyse the importance of features using boosting methods. Figure 6 plots the importance of features of the Tejo's learning model, where the sum of all features importances is equal to one. Figure 6a shows the 10 most-important features for anomaly detection in VoltDB serving Voter, seven out of 10 corresponding to performance counters of TCP layer of VMs. This suggests that the peer-to-peer communication pattern among the VoltDB cluster is key for anomaly detection. To provide insights into all 147 features, we organized them into seven distinct categories and measured their grouped importance, as depicted in Figure 6b. Indeed, it confirms that features from TCP performance counters form the main category, accounting for more than half the total of importance (0.5345). Surprisingly, the category of VoltDB features had the lowest importance for the anomaly detection task. This suggests that the contribution of database-specific features is negligible, therefore our learning model is likely to have similar predictive performance with different NewSQL databases. Results for TPC-C workload showed a similar trend.



(a) The 10 most-important features.

(b) Importance of features categories.

Fig. 6: Analysis of the importance of features for anomaly detection.

5 Related Work

Fault tolerance in distributed databases. Distributed databases use replication and advanced request scheduling to improve data availability. Bayou [18] is data storage that relies on replication to ensure data availability against fail-stop failures, but they are not able to deal with performance anomalies. Skute [1] provides an adaptive replication scheme that mitigates the impact of performance anomalies. However, it does not provide mechanisms to ensure high data availability, such as high throughput and bounded latency. Emerging cloud databases, like VoltDB and MongoDB, offer high data availability using enhanced main memory data structures [22]. But, our findings of this and previous study [21] show that performance anomalies on the cloud, including malfunctioning network cards, disk and main memory, can undermine the performance of cloud databases.

Cake [25] offers a scheduling scheme to enforce high-level data availability requirements for end users. However, Cake was not designed to identify faulty VMs. Eriksson *et al.* [9] provide a routing framework that helps cloud operators to mitigate the impact of network failures. We believe that our work is complementary

to theirs. Alerts from Tejo about anomalies in network, memory, disk and CPU of VMs, can contribute to enhance the efficiency of such scheduling mechanisms.

Anomaly detection with statistical learning. Anomaly detection is commonly implemented based on an *unsupervised learning method*. Gujrati *et al.* [13] provide prediction models based on event logs of supercomputers to detect platform-wide anomalies, whereas we are interested in detecting anomalous VMs based on monitoring data. Chen *et al.* [6] propose an anomaly detection approach for large-scale systems that improves the prediction efficiency of an entropy-based information theory technique by performing a principal component analysis (PCA) of system inputs. However, this introduces computational overhead that undermines its scalability and causes a slowdown in anomaly predictions. While we focus on detecting performance anomalies in NewSQL databases, Lan *et al.* [14] provide a general-purpose anomaly detection approach that relies on features selection to enhance prediction efficiency. Similarly, Guan and Fu [12] perform feature extraction based on PCA to identify the most relevant inputs for anomaly detection. Yet, results of our previous work [21] confirm that a supervised method with all features outperforms a unsupervised one by reducing the number of false positives by 10%. In this work, we extended our previous supervised learning model to detect multiple classes of anomalies based on SLO metrics.
Guan *et al.* [11] implement a probabilistic prediction model based on a *supervised learning method*. Although their model allows us to compare the dependability of virtualized and non-virtualized cloud systems, it suffers from poor prediction efficiency when it is used to predict cloud performance anomalies. Tan *et al.* [24] propose general-purpose prediction model to prevent performance anomalies. Their *supervised learning*-based model combines 2-dependent Markov chain model with the tree-augmented Bayesian networks. But, the authors did not provide information about the prediction efficiency and the capacity of their approach to generalize. We show with Tejo that the choice of the learning algorithm and features contribute to enhance predictive efficiency of performance anomalies.

6 Conclusion

The emerging stream processing platforms rely on NewSQL databases deployed on the cloud to compute big data with high velocity. However, performance anomalies caused by faults on the cloud infrastructure, that are likely to be common, may undermine the capacity of NewSQL databases to handle fast data processing. To analyse these performance anomalies, we proposed Tejo, a supervised anomaly detection scheme for NewSQL databases. This scheme allows us to evaluate the performance of NewSQL database as faults on network, memory, CPU, and disk occur. Experiments with VoltDB, a prominent NewSQL database, showed that the 99^{th} percentile latency soars two orders of magnitude as memory and network faults happen. We showed that Tejo also provides a learning model to detect these performance anomalies. Our findings suggest that learning algorithms based on boosting methods are better to detect anomalies on a VoltDB cluster, and features from the TCP layer of VMs are the best predictors. Results also suggest that the contribution of VoltDB-specific features is negligible, therefore our learning model is likely to have similar efficiency with different NewSQL databases.

Acknowledgments

This research is partially funded by the project Secured Virtual Cloud (SVC) of the French program Investissements d'Avenir on Cloud Computing.

References

1. N. Bonvin, T. G. Papaioannou, and K. Aberer. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In *SoCC*, 2010.
2. L. Breiman. Random forests. *Machine learning*, 2001.
3. M. Carbone and L. Rizzo. Dummynet revisited. *ACM SIGCOMM*, 2010.
4. U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, J. Meehan, A. Pavlo, M. Stonebraker, E. Sutherland, N. Tatbul, et al. S-store: A streaming newsql system for big velocity applications. *VLDB*, 2014.
5. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM CSUR*, 2009.
6. H. Chen, G. Jiang, and K. Yoshihira. Failure detection in large-scale internet services by principal subspace mapping. *TKDE*, 2007.
7. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 1995.
8. T. Do, M. Hao, T. Leesatapornwongsa, T. Patana-anake, and H. S. Gunawi. Limplock: Understanding the impact of limpware on scale-out cloud systems. In *SoCC*, 2013.
9. B. Eriksson, R. Durairajan, and P. Barford. Riskroute: a framework for mitigating network outage threats. In *CoNEXT*, 2013.
10. J. H. Friedman. Recent advances in predictive (machine) learning. *Journal of classification*, 2006.
11. Q. Guan, C.-C. Chiu, and S. Fu. Cda: A cloud dependability analysis framework for characterizing system dependability in cloud computing infrastructures. In *PRDC*, 2012.
12. Q. Guan and S. Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *SRDS*, 2013.
13. P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White. A meta-learning failure predictor for blue gene/l systems. In *ICPP*, 2007.
14. Z. Lan, Z. Zheng, and Y. Li. Toward automated anomaly identification in large-scale systems. *TPDS*, 2010.
15. J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.
16. B. C. Love. Comparing supervised and unsupervised category learning. *Psychonomic Bulletin & Review*, 2002.
17. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *JMLR*, 2011.
18. K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: replicated database services for world-wide applications. In *ACM SIGOPS*, 1996.
19. K. Ren, A. Thomson, and D. J. Abadi. Lightweight locking for main memory database systems. In *VLDB*, 2012.
20. E. Schurman and J. Brutlag. The user and the business impact of server delays, additional bytes, and http chunking in web search, 2009.
21. G. Silvestre, C. Sauvanaud, M. Kaâniche, and K. Kanoun. An anomaly detection approach for scale-out storage systems. In *IEEE SBAC-PAD*, 2014.

22. M. Stonebraker. Sql databases v. nosql databases. *ACM Communications*, 2010.
23. M. Stonebraker and A. Weisberg. The voltdb main memory dbms. *Data Engineering*, 2013.
24. Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *ICDCS*, 2012.
25. A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. Cake: enabling high-level slos on shared storage systems. In *SoCC*, 2012.