AISLE: Assessment of Provisioned Service Levels in Public IaaS-Based Database Systems

Jörn Kuhlenkamp(✉), Kevin Rudolph, and David Bermbach

Information Systems Engineering Research Group,
Technische Universität Berlin, Berlin, Germany
{jk,kr,db}@ise.tu-berlin.de

Abstract. When database systems running on top of public cloud services run into performance problems, it is hard to identify the concrete infrastructure service for which provisioning additional resources would solve said performance problem. In this work, we present AISLE, which develops a model for expected service levels and includes metrics which assess values from service level monitoring to identify these cloud services. Using AISLE, we develop such a model for the Amazon EBS service and evaluate our approach in experiments with Apache Cassandra running on top of EBS-backed EC2 instances.

Keywords: Cloud computing · IaaS · Service levels · Cloud monitoring

1 Introduction

Today's database systems often run on top of IaaS cloud services where the database administrator provisions the required amount of resources for each underlying service. A systematic approach for identifying the required amount is still missing today so that the administrator will typically provision resources based on his gut feeling and adapt later on. Even if the database system is already deployed and running and even if detailed monitoring data is available, it is challenging to identify the specific cloud services where additional resources should be provisioned. This is typically the case since cloud providers offer only very limited Service Level Agreements (SLAs) so that it is hard to interpret monitoring results.

Another problem is that not only the workloads, which the database system is confronted with, change over time requiring adaption of provisioned resource amounts but also the observable service levels of cloud services may vary over time due to a lack of SLAs, e.g., [3].

In this work, we present AISLE (**A**ssessing **I**nfrastructure **S**ervice **L**evel **E**nvironments), an approach that develops a model for an Expected Service Level (ESL) based on SLA information, documentation, and benchmarking results. Part of this approach are three metrics which enable the database administrator to easily assess whether the observed monitoring data for a specific service quality is limited by this expected service level, thus, identifying cloud services

A. Barros et al. (Eds.): ICSOC 2015, LNCS 9435, pp. 154–168, 2015.
DOI: 10.1007/978-3-662-48616-0_10

where additional provisioned resources are likely to positively affect the database system. We then use this approach to develop an ESL model for the Amazon EBS service[1] based on documentation and benchmarking. As an evaluation, we deployed an Apache Cassandra cluster on EBS-backed EC2[2] instances and verified experimentally that provisioning extra resources for the services identified by our metrics yields much better results than other more intuitive scaling approaches.

The paper is structured as follows: Sect. 2 gives a brief introduction to selected cloud services, systems and benchmarking tools. Section 3 describes AISLE, and Sect. 4 presents our application of AISLE to the EBS service. In Sect. 5, we use the model developed in Sect. 4 to evaluate our approach before discussing related work (Sect. 6) and coming to a conclusion (Sect. 7).

2 Background

In this section, we will briefly introduce cloud services, systems, and tools which we will refer to later.

Amazon Web Services: The *Amazon EC2* service offers virtual machines (*EC2 instances*). EC2 instances have an instance type which determines the amount of resources, e.g., virtual CPU cores or memory, available to applications running on that instance. For disk storage, EC2 instances can use either ephemeral disk or EBS volumes, instances of the *Amazon EBS* service. EBS volumes come in different sizes and have a type (e.g., *Provioned IOPS*) which determines the service level and the amount of dedicated bandwidth.

Apache Cassandra: Cassandra, a NoSQL system initially developed as a write-efficient database for the Facebook message inbox [10], is maintained as Apache project and widely used, e.g., by Netflix. Cassandra implements the Dynamo replication architecture [7] and a BigTable-inspired schemaless data model and storage engine [4].

Flexible I/O Tester: The Flexible I/O Tester (FIO)[3] is a tool for studying disk or network performance metrics on Linux systems. FIO works by synthetically creating workloads based on a stochastic model and offers extensive configuration options which makes it particularly suitable for our needs.

Yahoo! Cloud Serving Benchmark: The Yahoo! Cloud Serving Benchmark (YCSB) [5] is an extensible, modular benchmarking tool which emulates clients of a distributed database system and measures client-side performance metrics. YCSB comes with a set of standard workloads of which we use workloads A and B; A is a write-heavy (50 % read and 50 % update requests), B a read-heavy (95 % reads and 5 % updates) workload. We use both workloads with a `uniform` request distribution.

[1] aws.amazon.com/ebs.
[2] aws.amazon.com/ec2.
[3] linux.die.net/man/1/fio.

3 Assessing Cloud Service Levels

In this section, we describe AISLE, which transforms monitoring data into a quality metric-independent score describing how close the specific metric is to an ESL. As cloud providers give very few guarantees in SLAs, we first discuss how a (potentially complex) model – describing service levels a cloud service user could realistically observe – can be developed. Afterwards, we introduce three metrics which describe how a given monitoring value ranks compared to said model. These metrics can then be used to easily identify cloud services where additional resources are likely to have a strong effect on the database system.

3.1 Deriving a Model for Expected Service Levels

Cloud providers rarely offer SLAs that characterize service levels a database administrator can expect from the cloud service(s) the database is running on. If any, these SLAs typically give very limited guarantees [2]. Beyond these *explicitly* guaranteed service levels, additional information can be gained from documentation or via infrastructure benchmarking – this information describes service levels that are not guaranteed but which can still be realistically expected in database deployments. Explicit SLAs will typically be very precise, guaranteeing concrete metric values, e.g., a compute service might guarantee a minimum number of 1.5 Tera Flops per virtual machine. Implicit SLAs, in contrast, could be based on large data sets describing relative frequencies of values measured, e.g., EC2 instances would, based on [12], have a distribution function with two peaks characterizing their computation power. Alternatively, a documentation-based implicit SLA could require a relatively complex model, as we will see later in this paper. In any case, a model for ESLs should describe relative frequencies of service levels which can be expected when using that specific cloud service, i.e., an ESL is only for the most simplistic scenario a single value but will typically be a distribution of values. Also, even the "best"[4] value of an ESL is not necessarily the "best" value which can be reached – positive outliers are always possible – but it describes service levels that can be realistically reached in practice.

3.2 Normalizing Monitoring Data

For a given Observed Service Level (OSL), which can be obtained through runtime monitoring, we would like to characterize how "close" this value is to the ESL of the corresponding service quality. For quality-independent description and analysis of this "closeness", we propose to normalize service levels in a way that assigns 100 % to observed service levels that reach exactly the ESL. We call this normalized service level *Service Level Pressure* (SLP). Please, note that SLP can reach values beyond 100 % whenever an observed service level exceeds the ESL.

[4] What "best" is, highly depends on the respective quality metric.

The SLP for a given OSL o is defined as the ratio of o to the ESL. If the ESL is a single value then SLP(o) is trivial to calculate (e.g., if ESL is 5 and o is 3, then SLP(o) is 60 %.). If the ESL is a discrete distribution assigning relative frequencies f to service level values s, then the SLP is calculated as the weighted sum of all ratios o/s, i.e.,

$$SLP(o) = \sum_{\forall s} \frac{o}{s} * f \tag{1}$$

As SLP values are likely to show a large variability over time and short random spikes rarely have an influence on the database system, we propose to introduce the concept of time into two new metrics which are built on top of SLP: For a given time series of SLP values, we are interested in the periods of time during which these SLP values exceed a threshold that the database administrator deems critical, e.g., 70 %. Specifically, we propose to use the following two metrics to describe these critical pressure durations:

- *Maximum Critical Pressure Time:* MCPT(c) describes the longest period of time during which the SLP values continuously exceeded the value c.
- *Total Critical Pressure Time:* TCPT(c) describes the total time that the SLP values of the time series exceeded the value c. TCPT(c) is expressed as percentage of the length of the time series.

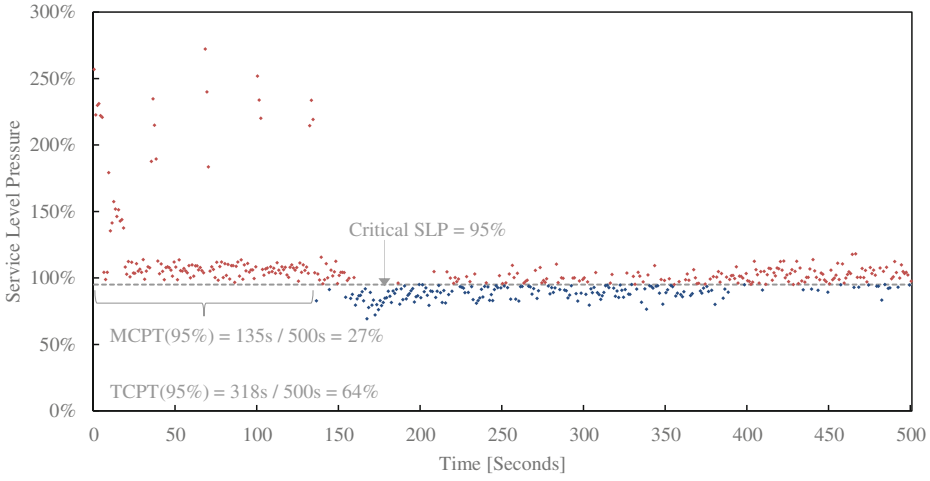


Fig. 1. Maximum critical pressure time and total critical pressure time based on a service level pressure threshold of 95 % for a 500 s interval.

Furthermore, beside using absolute values, we express both metrics as relative durations regarding the overall duration of the analyzed monitoring period in percent. Figure 1 shows an example of MCPT and TCPT. High MCPT values

imply a critical impact, e.g., caused by log consolidation [1] or online data migration [9], on the database system so that the amount of resources provisioned from the corresponding cloud service should be reconsidered. High TCPT values, on the other hand, are caused by the number of requests that the database receives, i.e., they are a hint that indicates from which cloud services the database administrator should provision more resources if the additional load persists.

4 Expected Service Level Model

In this section, we present an ESL model for the Amazon EBS service to apply AISLE to an *infrastructure topology* (topology) that includes EBS volumes. First, we present an analytical ESL model based on the EBS documentation. Second, we present results for an extended experimental evaluation of EBS, our experimental toolkit FIOEBS and an resulting experimental ESL model. Third, we discuss our results and compare analytical and experimental ESL models.

4.1 Analytical Expected Service Level Model

In implicit SLs, Amazon EBS gives throughput guarantees for volumes and connections according to the documentation for the EBS API version 2015-03-01. Precisely, volume throughput is quantified in (i) *normalized operations per second* (no/s) and (ii) bandwidth per second (bw/s). Furthermore, connection throughput is quantified in bw/s. A *normalized operation* (NO) $no(o)$ describes an operation o that is issued against a volume as a function of the operation size in KiB d_o. Therefore, a single operation is divided into a number of NOs based on a *normalized operation size* nd_o. If not stated otherwise, we assume a normalized operation size of $nd_o = 256$ KiB. Therefore, a single operation equals no NOs: $no(o) = \lceil d_o / nd_o \rceil$. For example, an operation $o1$ with request size $d_{o1} = 512$ KiB implies two NO.

An IO volume is provisioned with a configurable number of *provisioned normalized operations* (PNO) per second $pno_{io} = 0, \ldots, 20000$. The maximum number of configurable PNO depends on the configured volume size in GiB vs_v of a volume v: $\max pno_{io}(v) = 30 * vs_v$. Furthermore, an IO volume provisions a guaranteed bw/s pbw_{io}: $pbw_{io} = 320$ MiB/s.

EBS-optimized connections provide a dedicated guaranteed bw/s in MiB $pbw_{opt} = 62.5, 93.75, 125, 250, 500$ that depends on the instance type. Non EBS-optimized connections do not provide any explicit performance SLs.

4.2 Experiment Setup

In this section, we present a subset of the benchmarking experiments we used while trying to break and to extend the analytical performance model.

Infrastructure: We conduct all presented experiments in the same EC2 availability zone of *eu-west-1* and use three different EC2 instance types in our

experiments: t2.micro, t2.small, m3.medium. In the tests presented, we used IO Volumes (io1) with sizes of 100 GiB and PNOs of 300, 900 and 1800.

Benchmarking: We use FIO in version 2.1.3 to generate workloads and collect measurements. We deviate from the standard configuration with the following parameters. We emulate synchronous requests, i.e., `ioengine=sync`, disable page cache access, i.e., `direct=1`, refill buffers for operations, i.e., `refill_buffers=1`, and invalidate the buffer-cache, i.e., `invalidate=1`. We generate a synthetic data set of up to 1 file and with a file size of 70 GiB. Before each experiment, we execute a *load phase* in which we create file descriptors and write every block of the complete dataset.

Workload: We use a *Ramp Phase* of 120 s in which no measurements are collected to avoid skewed measurements during initialization. Afterwards, we execute a *Run Phase* for 120 s during which we collect measurements. We use a single thread to issue operations with an operation size of 4, 32, 64, 256 or 1024 KiB. We issue operations with a random variation of the file offset between issued operations. Furthermore, we use 5 workload mixes with different combinations of read percentage (R) and write percentage (W): (i) R100/W0, (ii) R75/W25, (iii) R50/W50 (iv) R25/W75 and R100/W0.

Measurements: We collect performance measurements with a granularity of 1 s on the user-side, i.e., Iostat, and 1 min on the provider-side, i.e., CloudWatch. Precisely, we measure throughput, i.e., average number of operations per second (ops/s) and average bandwidth per second (bw/s) in KiB, and, latency, i.e., average per operation latency. Due to space restrictions, latency measurements are not presented in this paper but included in the primary data set.

Implementation of the Experiment Toolkit: We designed and implemented an extensible toolkit called *AisleEbs*[5] to automate the setup of testbeds and the execution of experiments. Furthermore, we want to foster reproducibility of our experiments. AisleEbs allows to setup, execute and aggregate results for a number of FIO experiments on Amazon EC2. We implemented AisleEbs based on the configuration management system Ansible[6]. Figure 2 provides a concise overview of the interactions of AisleEbs.

Experiments: Due to restricted space, we only include a subset of our experiments and datasets. Within this paper, variable parameters in experiments are: instance, instance type, PNOs, workload mix and operation size. The presented results correspond to a total of 1125 experiments that are composed as follows: 1125 experiments = 3 (instance type) × 3 (PNOs) × 5 (operation size) × 5 (workload mix) × 5 (repetitions).

4.3 Experiment Results

Over all experiments, we do not see significant variations for the three parameters instance, instance type and workload mix. Therefore, we present results for

[5] github.com/jkuhlenkamp/aisle_ebs.
[6] ansible.com/home.

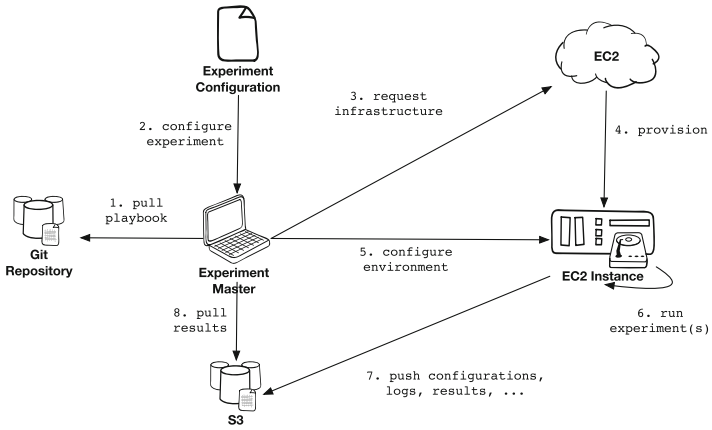
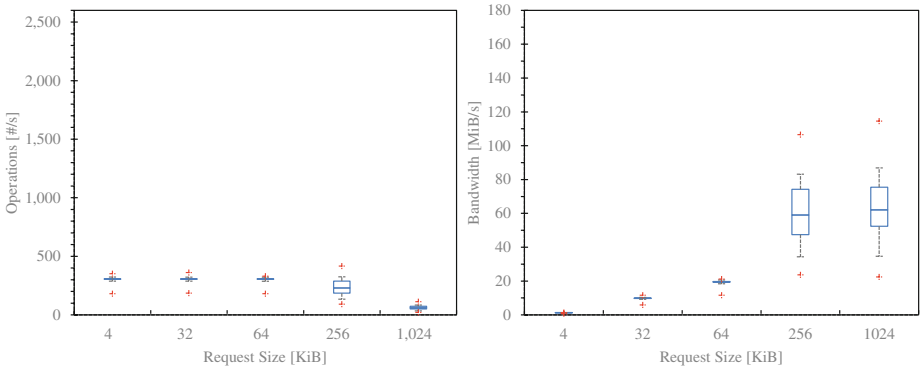


Fig. 2. Overview on interactions within a AisleEbs experiment testbed.

different PNOs and record sizes, i.e., 75 experiments per boxplot. We encourage the interested reader to analyze and/or reproduce our dataset that is available with our toolkit implementation. Presented Boxplots show averages instead of the medians, boxes present 25th and 75th percentils and whiskers 1st and 99th percentils.



(a) Operations per second

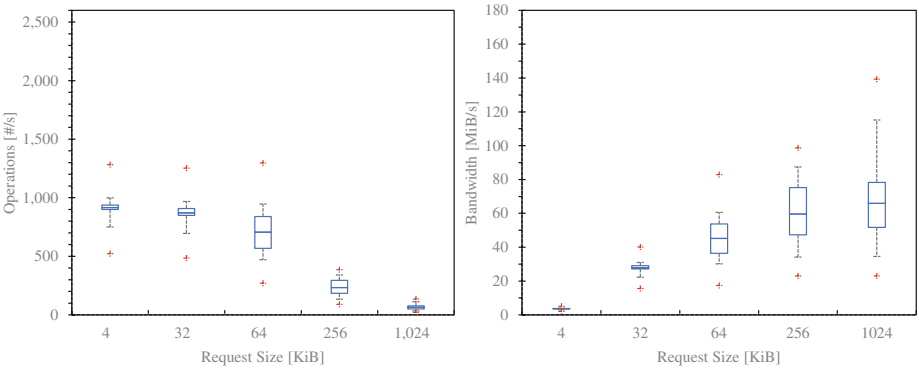
(b) Bandwidth per second

Fig. 3. Throughput for 100 GiB volume size and 300 PNOs.

300 Provisioned Normalized Operations. Figure 3 shows our results for 300 PNOs. Figure 3a shows an avg. throughput of 300 ops/s for operation sizes up to 64 KiB with low variability. With increasing operation sizes avg. ops/s drop down to a minimum of 60 ops/s for 1024 KiB. Figure 3b shows an proportional

increase in avg. bw/s with increasing operation sizes up to 64 KiB with low variability. For operation sizes of 256 KiB and 1024 KiB observed bw/s further increases by a lower factor at large variability.

Our results indicate that up to an operation size of 64 KiB throughput is limited by the number of PNOs. Furthermore, low variability indicates that PNO-based SLs are enforced effectively with low provider-side over- and under-provisioning. For request sizes of 256 KiB and 1024 KiB, our results indicate that throughput is limited by available bw/s. Furthermore, increasing variability indicates that bandwidth-based SLs are not enforced as strictly and/or effectively as operation-based SLs. We argue that the decreasing variability of measured ops/s for operation sizes of 1024 results from the fact that EBS enforces PNOs based on normalized ops/s, and we measure not normalized ops/s as perceived by FIOEBS.

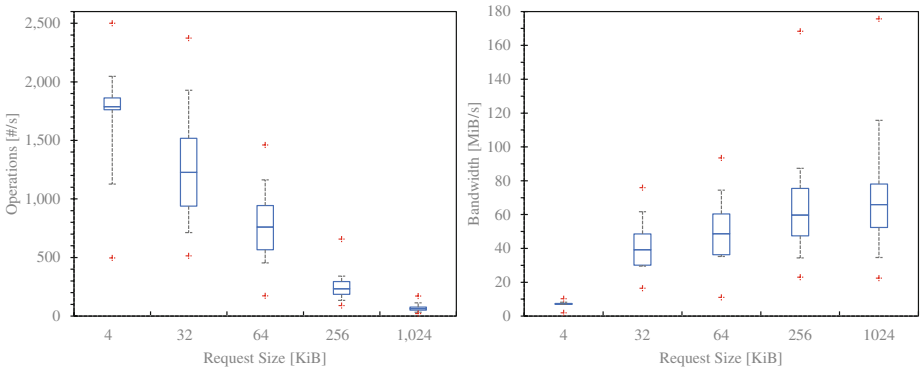


(a) Operations per second (b) Bandwidth per second

Fig. 4. Throughput for 100 GiB volume size and 900 PNOs.

900 Provisioned Normalized Operations. Figure 4 shows our results for 900 PNOs. Figure 4a shows an avg. throughput of 916 and 871 ops/s with low variability for operations sizes of 4 and 32 KiB. For larger operation sizes, we measure a large decrease of avg. ops/s. Figure 4b shows that proportional bandwidth increase stops and variability increases at operation sizes of 64 KiB. We observe a maximum bandwidth of 139 MiB/s for operations sizes of 1024 KiB. Our results indicate that up to a operation size of 32 KiB throughput is limited by the number of PNOs and for large operation sizes by bandwidth.

1800 Provisioned Normalized Operations. Figure 5 shows our results for 1800 PNOs. Figure 5a shows an avg. throughput of 1788 ops/s for operations sizes of 4 KiB. For larger operation sizes, avg. ops/s drop to 64 for 1024 KiB. Figure 5b shows that proportional bandwidth increase stops and variability increases for operation sizes larger than 4 KiB.



(a) Operations per second (b) Bandwidth per second

Fig. 5. Throughput for 100 GiB volume size and 1800 PNOs.

Our results indicate that up to a operation size of 4 KiB throughput is limited by the number of PNOs and for large operation sizes by bandwidth. In comparison to 900 PNOs, absolute throughput and throughput variability does not change significantly for operation sizes of 256 KiB and 1024 KiB.

4.4 Discussion

Our results indicate that an increase of PNOs for an volume does not increase avg. bw/s but maximum bw/s for that volume. Furthermore, variability at maximum throughput increases with increasing absolute throughput. We observe the highest throughput variability for the same PNOs at the turning points between operation-based and bandwith-based throughput limitation. If not subject to an operation-based limitation, we observe an avg. bw/s of 60 MiB for 256 KiB and 66 MiB for 1024 KiB, since indicating that t2.micro, t2.small and m3.medium instances share similar instance to EBS connection limitations. We argue that this observed behavior might change for other instance types and EBS-optimized volume connections, respectively.

For EBS volumes with small numbers of PNOs under workloads with small operation sizes, our results indicate that the analytical ESL model obtained from the EBS documentation predicts ESLs with high accuracy. However, for larger operation sizes and PNOs the accuracy of the analytical model quickly decreases. Overall, we argue that the documentation of ESLs without experimental evaluation is not sufficient to serve as an accurate information base for infrastructure optimization decisions. Experimental evaluation of public infrastructure services increases the overhead for applying AISLE. However, we argue that experimental ESL models can be reused for the evaluation of different deployments and shared between different IaaS users.

Next, we apply AISLE based on the obtained ESL model for EBS in an evaluation and optimization of a Cassandra deployment on EC2.

5 Use Case: Cassandra

In this section, we apply our method and obtained ESL model to a Cassandra deployment on two topologies.

5.1 Deployment Enviroment

We deployed Cassandra clusters with standard configuration parameters and no replication.

Infrastructure: For each experiemnt, we provision a 3-node topology of either m3.large or m3.medium EC2 instances, all in the same availability zone of the eu-west-1 region. Both instance types provide moderate networking performance and are not EBS-optimized. We attached a single 100 GiB IO volume with 300 PNOs to each instance.

Workload: As a load generator, we deployed YCSB on a single m3.large instance to emulate 100 client threads running for 30 min. Furthermore, we use for both setups m3.large and m3.medium initial datasets that exceed each instance's memory almost by a factor three. We emulate YCSB standard workloads A and B.

Toolkit and Measurements: We implemented the tool *AisleCassandra*[7] to automate experiment execution. We use *Amazon CloudWatch*[8] to obtain provider-side and *Iostat*[9] to obtain user-side monitoring measurements.

5.2 AISLE Application

We apply AISLE within our two topology setups under workloads A and B. First, we conduct an a priori analysis of the initial topologies. Second, we adjust the initial topologies and conduct a second a posteriori analysis. For both analyses, we present boxplots of SLP for ops/s (P_{op}) and SLP for bw/s (P_{bw}).

300 Provisioned Normalized Operations. Figure 6 shows our results for setups m3.medium and m3.large for workload A and workload B. For both setups and workloads, our results show a high P_{op} compared to a low P_{bw}. m3.large under both workloads and m3.medium under workload B show a P_{op} above 80 % on avg. and 92 % for the 75th percentile for all volumes except volume 1 in m3.medium. Since, indicating underprovisioned SLs regarding ops/s SLs. m3.medium under workload A shows 99th percentile P_{op} of less than 85 % for volume 1 and 3, since, results indicate small potential for optimization for both volumes. We adjust the volumes in both setups regarding ops/s SLs. Precisely, we increase PNOs from 300 to 1800 for all IO volumes.

[7] github.com/jkuhlenkamp/aisle_cassandra.
[8] aws.amazon.com/cloudwatch/.
[9] linux.die.net/man/1/iostat.

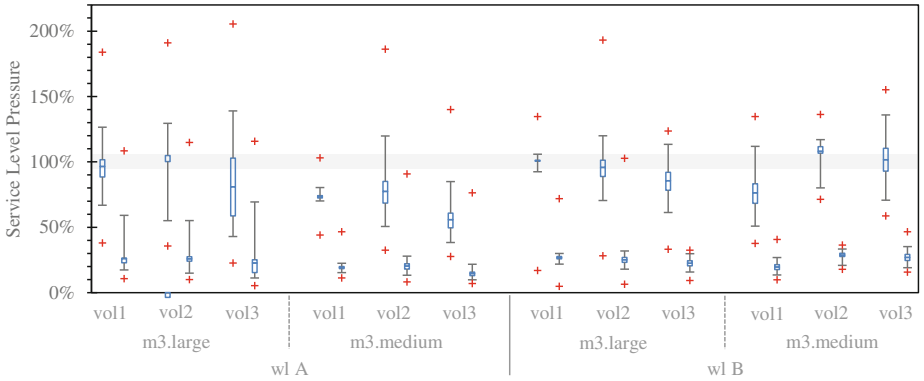


Fig. 6. Boxplots of Service Level Pressures for (i) ops/s and (ii) bw/s for EBS volumes in 3 node clusters and 300 Provisioned Normalized Operations.

1800 Provisioned Normalized Operations. Figure 7 shows our results for the adjusted setups m3.medium and m3.large for workload A and workload B. For both workloads and adjusted setups, our results show a shift from a high P_{op} to high P_{bw} in comparision to the inital setups. For both workloads, all volumes in m3.large report 25th percentile P_{bw} above 89 %. Therefore, our results indicate underprovisioned SLs regarding bw/s. For both workloads, all volumes in m3.medium report 99th percentile values for P_{op} and P_{bw} of less than 71 %. Therefore, indicating overprovisioned volumes regarding both SLs.

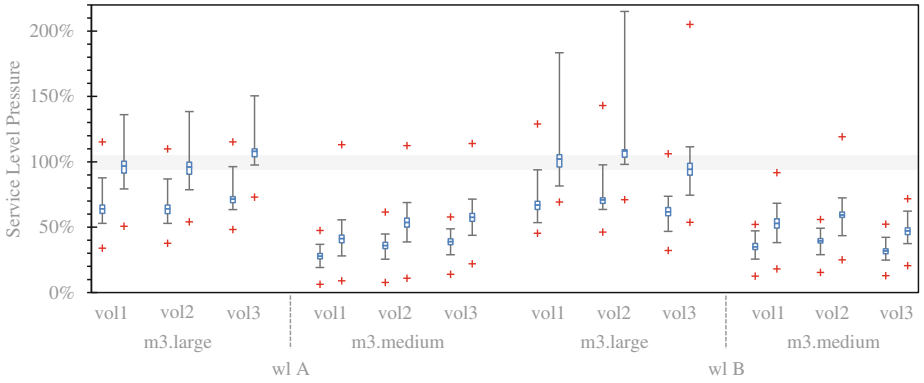


Fig. 7. Boxplots of Service Level Pressures for (i) ops/s and (ii) bw/s for EBS volumes in 3 node clusters and 1800 Provisioned Normalized Operations.

5.3 Discussion

We showed that AISLE is suitable to characterize the optimization potential regarding SLs of public IaaS-based topologies. However, we are aware that our approach might imply a certain implementation and management overhead in comparison to a trial and error-based scale-out strategy. The overhead is due to obtaining ESL models, the setup of monitoring infrastructure and conducting analysis. Therefore, we compare AISLE to a scale-out strategy. Furthermore, we discuss the adoption of provider-side monitoring solutions to decrease required management overheads for monitoring solutions.

AISLE vs. Scale-Out. The application of AISLE implies an additional management overhead for a user of an infrastructure service. Therefore, we compare AISLE to a trivial scale-out strategy, i.e., the provisioning of additional servers with the same configuration into an existing cluster. We refer to AISLE-based scaling as *Option A* (Experimental) and scale-out-based scaling as *Option B* (Analytical). We compare both approaches in terms of application SLs for the Cassandra cluster using a ratio for simpler comparison. Precisely, the ratio consists of the overall topology cost in $ per month divided by Cassandra throughput, i.e. avgerage client requests per second (req/s). For Option B, we assume a linear increase in throughput as indicated by previous research [9,14]. Furthermore, we select the corresponding cluster size for Option B as follows: We select the highest resulting throughput that is lower than the corresponding throughput for Option A. Table 1 shows a concise comparison between Option A and Option B. As shown in column *Opt. A - Opt. B* and indicated by negative comparison values, a AISLE-based scaling strategy results in a higher scaleup under lower costs per month for all workloads (wl A and wl B) and instances types and, as expected, a higher benefit for m3.large.

Table 1. Performance and costs comparison for scaling an initial Cassandra cluster (Base) with scaling strategy AISLE (Opt.A) vs. horizontal scale-out (Opt.B).

	Opt. A (Experimental)		Opt. B (Analytical)		Opt. A - Opt. B
	PIOPS = 1,800		PIOPS = 300		Savings
	#nodes	$/(req/s)	#nodes	$/(req/s)	$/(req/s)
m3.large, wl A (update-heavy)	3	0.08	14	0.22	−0.14
m3.large, wl B (read-heavy)	3	0.13	13	0.33	−0.20
m3.medium, wl A (update-heavy)	3	0.11	7	0.13	−0.02
m3.medium, wl B (read-heavy)	3	0.19	7	0.23	−0.04

Provider-Side vs. Client-Side Monitoring. AISLE requires monitoring of infrastructure resources. Therefore, we compare two common options that are available to a user to enable a monitoring solution: (i) client-side monitoring, i.e., Iostat, and (ii) provider-side monitoring, i.e., Amazon CloudWatch. Client-side monitoring implies a management and performance overhead. Still, client-side monitoring implies the freedom to adapt and verify monitored metrics and granularity. Provider-side monitoring implies a low management and performance overhead. However, users have limited control over the monitoring solution and must trust a provider.

Next, we apply AISLE with monitoring traces obtained from Amazon CloudWatch. According to the latest CloudWatch documentation, i.e., API version 2010-08-01, CloudWatch provides a metric `VolumeConsumedReadWriteOps` that reports no/s based on a normalized operation size of 256 KiB with 1 min granularity. Figure 8 shows no/s reported by CloudWatch and Iostat for a single volume. Our measurements indicate that CloudWatch reports much higher no/s for the same 1 min interval. We further analyzed this behavior. We believe that CloudWatch reports no/s based on a normalized operation size of 16 KiB instead of 256 KiB. Therefore, we derived no/s for normalized operation size of 16 KiB for Iostat measurements. Figure 8 shows the derived scaling factor and the resulting no/s model for a normalized operation size of 16 KiB for Iostat. Visual analysis indicates that this is indeed the case.

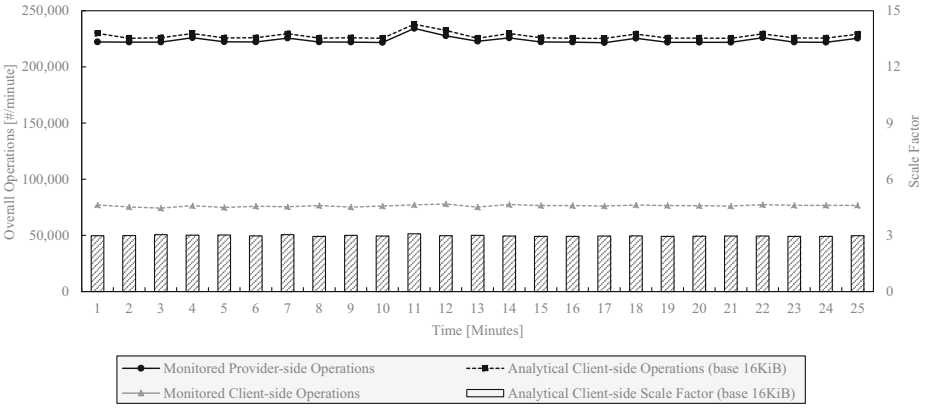


Fig. 8. Comparison of client-side (Iostat) and provider-side (CloudWatch) throughput measurements for different normalized operation sizes.

6 Related Work

There is no directly related approach with the same scope as AISLE. Existing publications fall in either of the following three groups:

Infrastructure Optimization: Several autonomous computing approaches adapt provisioned resources at runtime for database systems, e.g., [6,13,17]. In contrast to AISLE, these approaches all require explicit knowledge on the implementation details of the database systems whereas AISLE is theoretically application-agnostic – even though only evaluated for database systems – and treats the system as a blackbox. As AISLE in its current version does not adapt provisioned resources but rather provides the necessary information to do so to an administrator, these approaches complement our work.

Provider Perspective: Approaches like [16,18] take a provider perspective and focus, e.g., on offering optimal physical resource sets to virtual machines depending on the runtime requirements of the virtual machine. AISLE, in contrast, takes the perspective of a client who does not have any influence and knowledge on the service internals.

Infrastructure Benchmarking: Performance benchmarking of infrastructure services like Amazon EBS has been done before, e.g., [8,11,15]. We introduce new metrics to not only measure but also interpret results. For the special case of Amazon EBS, we also ran our experiments in more configuration setups and could, thus, also observe more fine-grained results: For instance, our experiments indicate low bandwidth variability for a certain combinations of EBS volume configurations and operation sizes whereas [11] reported overall high bandwidth variability.

7 Conclusion

In this work, we presented AISLE, an approach that develops a model for an Expected Service Level based on SLA information, documentation, and benchmarking results. Part of this approach are three metrics which enable the database administrator to easily assess whether the observed monitoring data for a specific service quality is limited by this expected service level, thus, identifying cloud services where additional provisioned resources are likely to positively affect the database system. As an example, we have based on AISLE developed an ESL node for the Amazon EBS service and verified experimentally that provisioning extra resources for the services identified by our metrics yields much better results than other more intuitive scaling approaches. We believe that our approach is not limited to database systems and will, therefore, in future work try to extend it to all kinds of applications running on top of cloud services.

Acknowledgments. We thank Amazon Web Services for a generous grant that enabled us to run our experiments.

References

1. Ahmad, M.Y., Kemme, B.: Compaction management in distributed key-value datastores. PVLDB **8**(8), 850–861 (2015)

2. Baset, S.A.: Cloud SLAs. ACM SIGOPS Operating Syst. Rev. **46**(2), 57 (2012)
3. Bermbach, D., Tai, S.: Benchmarking eventual consistency: Lessons learned from long-term experimental studies. In: Proceedings of the 2nd International Conference on Cloud Engineering (IC2E). IEEE (2014)
4. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. **26**(2), 1–26 (2008)
5. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing
6. Copil, G., Trihinas, D., Truong, H.-L., Moldovan, D., Pallis, G., Dustdar, S., Dikaiakos, M.: ADVISE – a framework for evaluating cloud service elasticity behavior. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 275–290. Springer, Heidelberg (2014)
7. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles
8. Dittrich, J., Quian, J.A.: Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc. VLDB Endow. **3**(1), 460–471 (2010)
9. Kuhlenkamp, J., Klems, M., Röss, O.: Benchmarking scalability and elasticity of distributed database systems. Proc. VLDB Endow. **7**(12), 1219–1230 (2014)
10. Lakshman, A., Malik, P.: Cassandra. ACM SIGOPS Operating Syst. Rev. **44**(2), 35–40 (2010)
11. Leitner, P., Cito, J.: Patterns in the chaos a study of performance variation and predictability in public IaaS clouds. In: Proceedings of the 24th International World Wide Web Conference (WWW 2015). ACM, Florence (2015)
12. Lenk, A., Menzel, M., Lipsky, J., Tai, S., Offermann, P.: What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. In: 2011 IEEE International Conference on Cloud Computing (CLOUD), pp. 484–491 (2011)
13. Lim, H.C., Babu, S., Chase, J., Parekh, S.: Automated control in cloud computing: challenges and opportunities. In: Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, pp. 13–18 (2009)
14. Rabl, T., Sadoghi, M., Jacobsen, H.A., Gómez-Villamor, S., Muntés-Mulero, V., Mankowskii, S.: Solving big data challenges for enterprise application performance management. Proc. VLDB Endow. **5**(12), 1724–1735 (2012)
15. Scheuner, J., Leitner, P., Cito, J., Gall, H.: Cloud WorkBench - infrastructure-as-code based cloud benchmarking. In: Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014) (2014)
16. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: CloudScale. In: Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC 2011, pp. 1–14 (2011)
17. Trushkowsky, B., Fox, A., Franklin, M.: The scads director: Scaling a distributed storage system under stringent performance requirements. In: Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST 2011), pp. 12–25. USENIX Association, Berkeley (2011)
18. Yao, J., Jung, G.: Bottleneck detection and solution recommendation for cloud-based multi-tier application. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 470–477. Springer, Heidelberg (2014)