AdeptDC's Incident Management Solution for AWS Services

Preamble: AdeptDC offers an online incident management solution for IT/IoT services which alarms incidents without any explicit safety thresholds and supervisory ground truths.

Abstract: In this document, a case study has been discussed to illustrate how AdeptDC can assist SREs and developers in managing incidents in AWS services. In particular, we will verify the hypothesis that the online incident management solution can detect incident even in the absence of explicit thresholds and historical ground truths for supervised learning.

info@adeptdc.com | www.adeptdc.com

Introduction: Modern applications run on a service-oriented architecture (SOA), comprised of a large number of microservices. For resilience, these services replicated across multiple zones and regions. The SOA architecture is characterized by its complexity and dynamicity. Often, different service components are intricately related and this relationship changes dynamically as the application load profiles change in time. This complexity and dynamicity make incident management particularly challenging because:

1. It is hard to set up an operating baseline for SOA-based services supporting dynamic workloads. Therefore, it is challenging to hardcode operating thresholds.
2. It is cost-prohibitive to keep an incident database with sufficient true positive incidents.
3. The nature of incidents depend of localized features of an environment. Therefore, it is hard to come up with a general-purpose database that can be shared across the SRE and developer community.
4. The speed of operating data is increasing so the incident management policy must be dynamically updated which is impossible for human operators.

This background leads us to our hypothesis that if we can add value to existing IT/IoT service managenement teams if we could build an incident management solution that can detect incident without explicit operating thresholds and supervisory ground truths. This document discusses a case study to validate the proposed hypothesis using a prototype software developed by AdeptDC.

Case Study: The hypothesis for this case study is AdeptDC's software can detect incidents in an unsupervised manner for AWS services. To test this hypothesis, we set up a test infrastructure, as shown in Figure 1, using AWS Elastic beanstalk. We define incidents as point anomalies i.e., the change in a metric value beyond a normal baseline and concept drift, i.e., the change in the normal baseline for a metric.

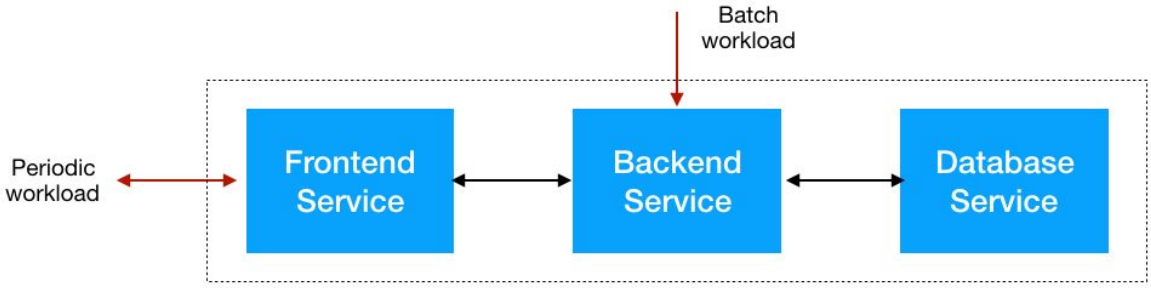

Figure 1: Test Setup for AdeptDC

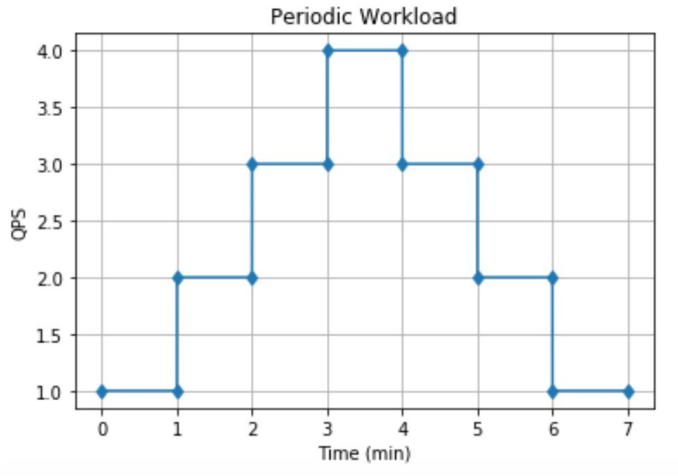

Figure 2: Periodic Workload on Frontend

The periodic workload on the Frontend Service simulates a user profile for the app. It is a continuous step workload with time period of 7 min, as shown in Figure 2.

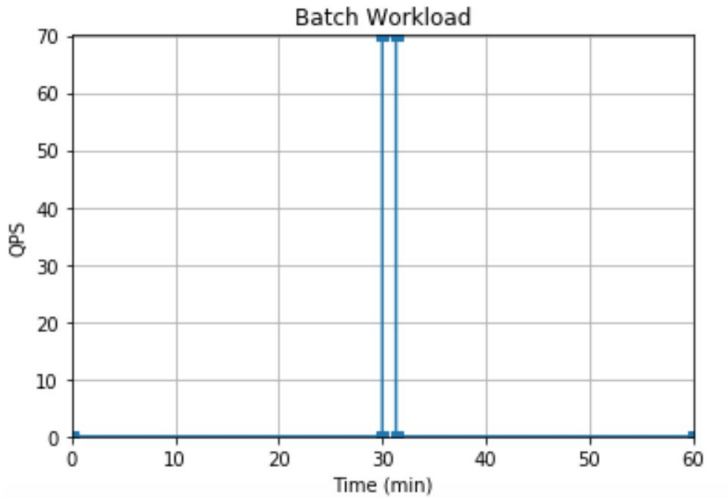

Figure 3: Batch Workload on Backend

Figure 3 shows randomized batch workload that varies between 60-80 QPS and lasts for 1-2 minutes. It happens at least once every three hour and every pulse is separated by at least one hour. The mathematical model for the batch workload is given by:

QPS **~(1- w1)*0 + (w1)*(60 + rand.random()*(80-60))**

Spike duration(min) **~ 1 + rand.random()*(2-1)**

Time interval between two successive pulses(min) ~ **60 + rand.random()*(180-60)**

where w1 are marginal probability of having a pulse. Clearly w1 is pretty low.

w1 = (1 + rand.random()*(2-1)) /(60 + rand.random()*(180-60))

The maximum values of w1 is equal to **(1/30)**, while the minimum value is **(1/180)**.

The state of every service can be defined by four different components, including {request rate 2xx, request rate 4xx, request rate 5xx, application latency P 99}.

Results and Discussion: The hypothesis we want to prove that the prototype of the unsupervised incident detection algorithm can detect incident with reasonable fidelity. There are 12 different test operating metrics in the test infrastructure. Each of the services--frontend, backend, and database--has 4 operating metrics. We will discuss the incidents in each of the services separately and then we follow up with a unifying bottleneck analysis.

Incident Detection for Frontend Service

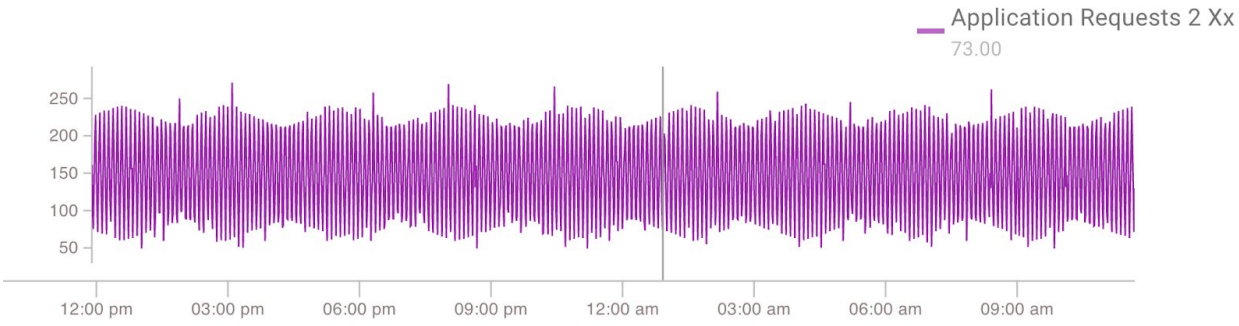

Figure 4: Incident Detection for Application Requests 2 Xx for the Frontend Service

As shown in Figure 4, the Application Request 2 Xx does not show any incident (no baseline shift, no point anomaly). It is quite expected as Frontend Requests 2 Xx periodically varies following a regular waveform pattern shown in Figure 2. It proves AdeptDC's solution does not gather false positives even with a rapidly fluctuating workload.

Figure 5: Incident Detection for Application Requests 4 Xx for the Frontend Service

As shown in Figure 5, the Application Request 4 Xx shows point anomalies (spikes). It indicates a possible bad request error. AWS documentation suggests a 400 error in Elastic Beanstalk could be coming from different possible issues as compiled in Table 1.

Figure 6: Incident Detection for Application Requests 5 Xx for the Frontend Service

Figure 6 shows zero incidents for 5 Xx application requests. It means no service internal error, such as: ServiceUnavailable, InternalFailure.

Table 1: AWS Elastic Beanstalk Common Errors with 4 Xx and 5 Xx Status Code (https://docs.aws.amazon.com/elasticbeanstalk/latest/api/CommonErrors.html)

Error Message	Description	HTTP Code
AccessDeniedException	Lack of sufficient access	400
IncompleteSignature	Lack of conformity to AWS standard	400
InvalidAction	Invalid operation requests	400

InvalidClientTokenId	Non-existent X.509 certificate or AWS access key ID	403
InvalidParameterCombination	Parameters that must not be used together are used together	400
InvalidParameterValue	An invalid or out-of-range value was supplied for the input parameter.	400
InvalidQueryParameter	The AWS query string is malformed or does not adhere to AWS standards	400
MalformedQueryString	The query string contains a syntax error	404
MissingAction	The request is missing an action or a required parameter	400
MissingAuthenticationToken	The request must contain either a valid (registered) AWS access key ID or X.509 certificate.	403
MissingParameter	A required parameter for the specified action is not supplied.	400
OptInRequired	The AWS access key ID needs a subscription for the service.	403
	The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.	400
ThrottlingException	The request was denied due to request throttling.	400
ValidationError	The input fails to satisfy the constraints specified by an AWS service.	400
ServiceUnavailable	The request has failed due to a temporary failure of the server.	503
InternalFailure	Unknown error, exception or failure	500

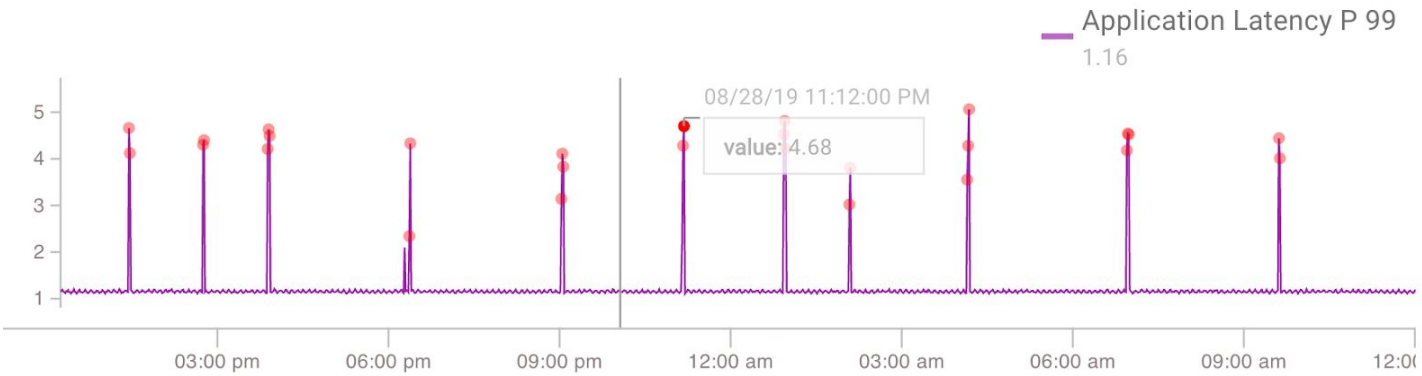

Figure 7: Incident Detection for Application Latency P 99 for the Frontend Service

Figure 7 shows the 24 incident distribution for Application Latency (in s) P99 for Frontend Service. It indicates spikes for 99 percentile application latency. One can infer the latency spikes are due to 400 error. While the number of latency incidents for last 24 hours is 26, that for 4 Xx is 9. It indicates a possibility of a deeper issue beyond just 4 Xx errors. Given the architecture as shown in Figure 1, the frontend latency can be related to other service latencies by following relationship:

Frontend Latency = Backend Latency + Database Latency + Network Latencies

Given all the services are residing on the same region, the network latency should be very small.

Incident Detection for Backend Service

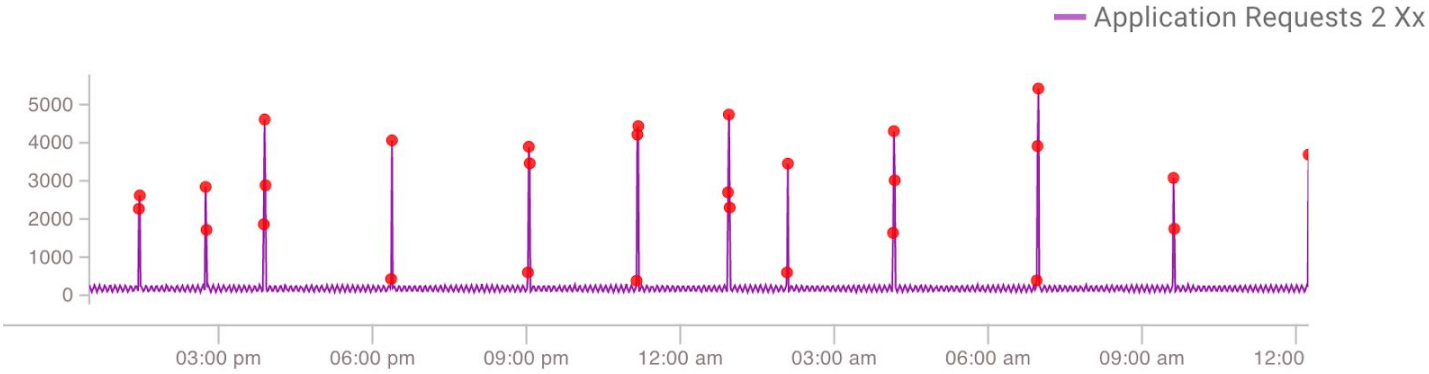

Figure 8: Incident Detection for Application Requests 2 Xx for the Backend Service

Figure 8 shows several incidents for Application Requests 2 Xx for the Backend Service. It indicates spikes for application requests. In fact there are 30 incidents in point anomalies in last 24 hours. It is interesting to note that incidents are happening in clusters and each cluster has 2-3 incidents. It can be related to the fact that the batch workload last for about 1-2 minutes and we are checking for incidents every minute. A deeper observation might reveal that the incident clusters of application request spikes are happening at a time interval of 1-3 hr. The time interval between two incident clusters is neither less than 1 hr nor greater than 3 hr. It can be related to the fact that batch workload is triggered only once per 1-3 hr. Another observation can be

made that the incident detection solution is robust to noise. It does not flag small fluctuations as incidents. It goes to show the high precision (fewer false positives) of AdeptDC's prototype algorithm.

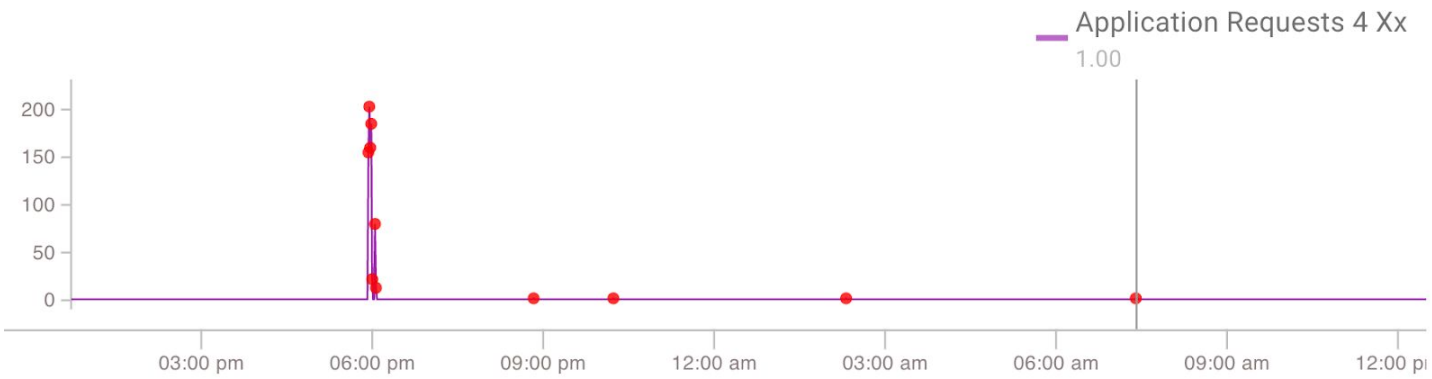

Figure 9: Incident Detection for Application Requests 4 Xx for the Backend Service

Figure 9 shows several incidents for Application Requests 4 Xx for the Backend Service. It indicates spikes for Bad Requests. In fact there are 11 incidents in the 24 hr test period. It is interesting to note that there are two distinct patterns. First there are four point anomalies at around 8:30pm, 10:15pm, 2:15pm, and 7:30am. There is only one error in each of these four cases; it could be due to some minor issues in the load balancer. But, there is a cluster of 7 incidents around 6pm, as shown in the following Table 2. This cluster indicates a possible degradation that requires operator attention. The possible scenarios include:

1. It is a bug which should not have been a 4 Xx but other internal errors.
2. A lot of invalid requests are sent by customers so this is expected. So not a bug but there are bad customers out there.
3. Client side code is not updated to a change in the server side. So fix the client side or roll back the server side.

Table 2: Backend 4 Xx Incident cluster captured around 08/28/19 6:00:00 PM (PST)

Time	Number of 4 Xx Application Requests
08/28/19 5:57:00 PM (PST)	154
08/28/19 5:58:00 PM (PST)	202
08/28/19 5:59:00 PM (PST)	159
08/28/19 6:00:00 PM (PST)	184
08/28/19 6:01:00 PM (PST)	21
08/28/19 6:04:00 PM (PST)	79

08/28/19 6:05:00 PM (PST)	12

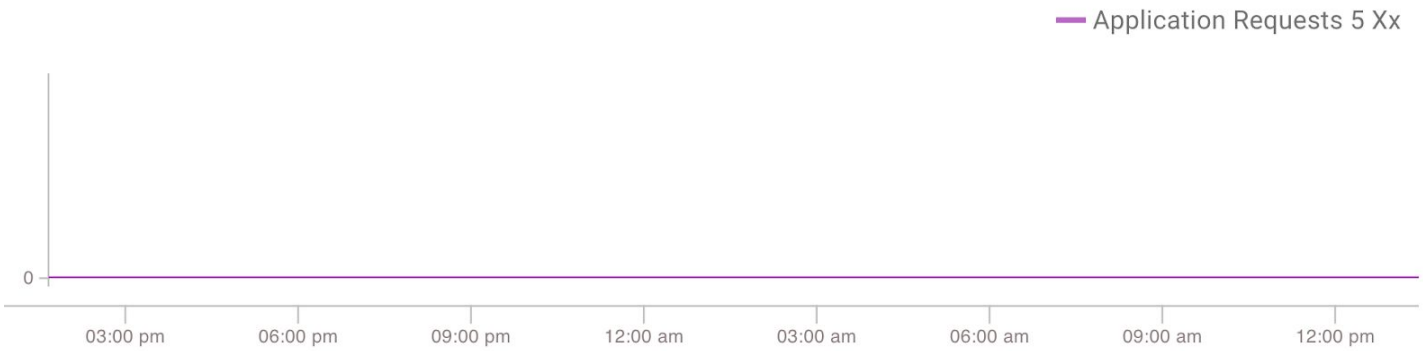

Figure 10: Incident Detection for Application Requests 5 Xx for the Backend Service

Figure 10 shows zero incidents for 5 Xx application requests for the Backend Service. It means no service internal error, such as: ServiceUnavailable, InternalFailure.

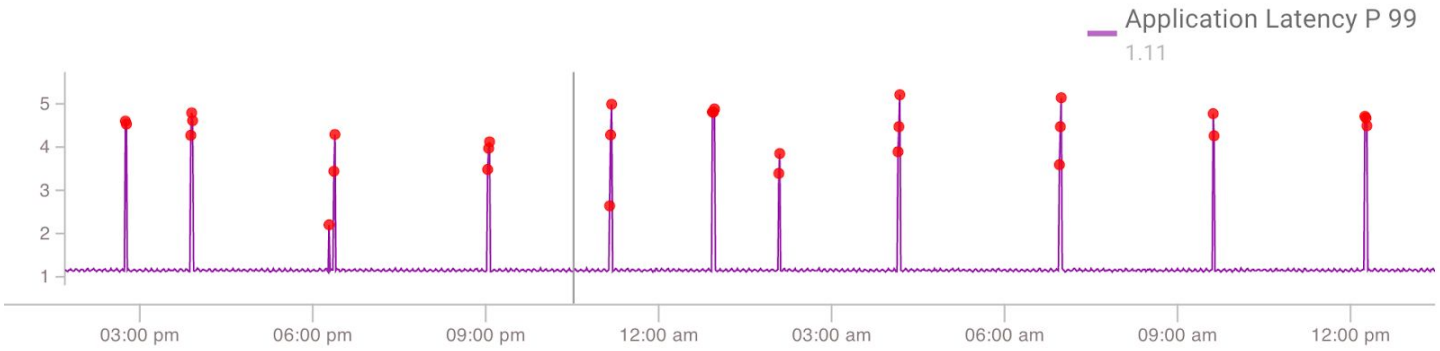

Figure 11: Incident Detection for Application Latency P 99 for the Backend Service

Figure 11 shows several incidents for Application Latency (in s) P99 requests. It indicates spikes for 99 percentile application latency. One can infer the latency spikes are due to 400 error. While the number of latency incidents for last 24 hours is 28, that for 4 Xx is 11. It indicates a possibility of a deeper problem. Given the architecture as shown in Figure 1, the backend latency can be related to other service latencies by following relationship:

Backend Latency = Database Latency + Network Latency

Given all the service components are residing on the same region, the network latency should be very small.

Incident Detection for Database Service

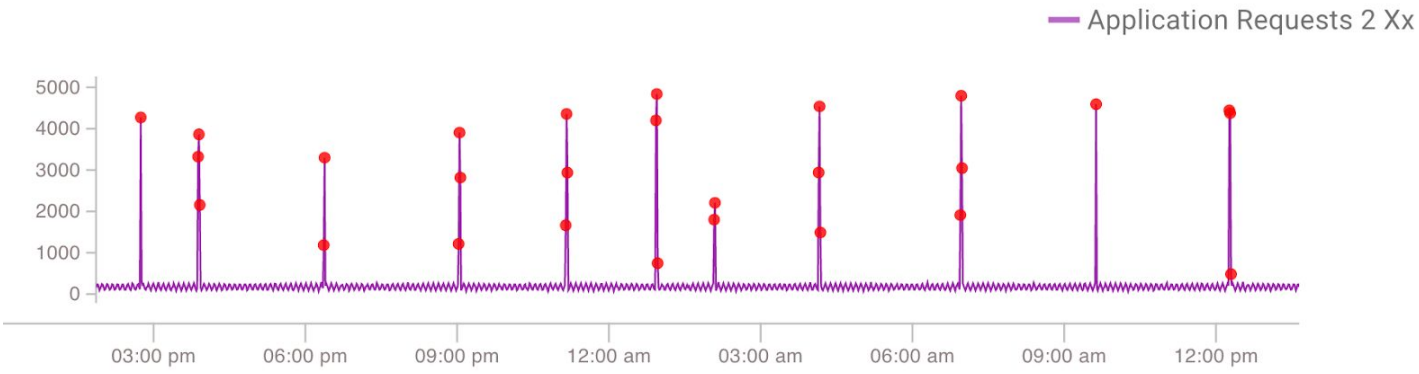

Figure 12: Incident Detection for Application Requests 2 Xx for the Database Service

Figure 12 shows there are 26 incidents in point anomalies in last 24 hours for Application Requests 2 Xx for the Database Service. As expected, Application Requests 2 Xx for the

Figure 13: Incident Detection for Application Requests 4 Xx for the Database Service

Figure 13 shows two incidents for Application Requests 4 Xx for the Database Service. There are two incidents: one, at around 8/28/2019 11:00pm and another at around 8/29/2019 1pm. These could be attributed to random infrastructure issues.

Figure 14: Incident Detection for Application Requests 5 Xx for the Database Service

Figure 14 shows zero incidents for 5 Xx application requests for the Database Service. It means no service internal error, such as: ServiceUnavailable, InternalFailure.

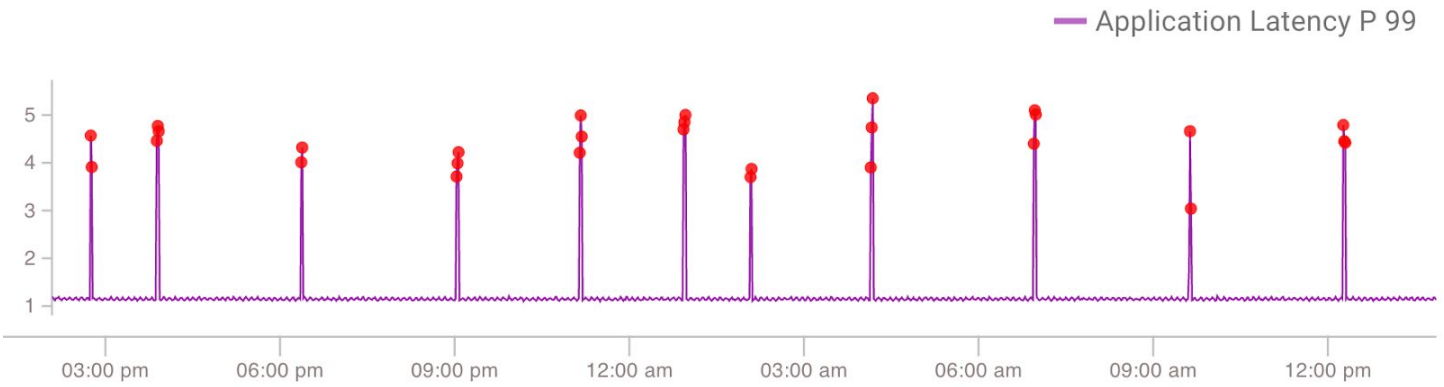

Figure 15: Incident Detection for Application Latency P 99 for the Database Service

Figure 15 shows several incidents for Application Latency (in s) P99 requests for the Database Service. It indicates latency spikes for 99% of the requests. One can cursorily infer the latency spikes are due to 400 error. While the number of latency incidents for last 24 hours is 31, that for 4 Xx is 2. It indicates a possibility of a deeper incident landscape. It can be noted that database latency spiking are occurring almost at the same time as database application request 2 Xx spikes, which in turn is concurrent to backend application request 2 Xx spikes.

Overall this subsection validates our fundamental hypothesis that AdeptDC's prototype can capture incidents in an operating metric in an unsupervised manner. In the next subsection (Bottleneck Analysis), we will discuss the context of incident detection and whether we could draw a systematic conclusion about the incident landscape from the detected incident flags.

Bottleneck Analysis

Table 3 shows the concurrencies of incidents for Backend Request, Database Latency, Backend Latency, and Frontend Latency. It goes to show the incidents are taking place almost at the same time barring few exceptions which can be attributed to minor infrastructure related issues.

Table 3: Concurrency Analysis for Incidents in Backend requests, Database latency, Backend latency, and Frontend latency

Backend Request	Database Latency	Backend Latency	Frontend Latency
8/28/2019 14:46:00	8/28/2019 14:46:00	8/28/2019 14:46:00	8/28/2019 14:46:00
8/28/2019 14:47:00	8/28/2019 14:47:00	8/28/2019 14:47:00	8/28/2019 14:47:00
8/28/2019 15:54:00	8/28/2019 15:54:00	8/28/2019 15:54:00	8/28/2019 15:54:00
8/28/2019 15:55:00	8/28/2019 15:55:00	8/28/2019 15:55:00	8/28/2019 15:55:00
8/28/2019 15:56:00	8/28/2019 15:56:00	8/28/2019 15:56:00	8/28/2019 15:56:00
No Incident	No Incident	8/28/2019 18:18:00	No Incident
8/28/2019 18:23:00	8/28/2019 18:23:00	8/28/2019 18:23:00	8/28/2019 18:23:00
8/28/2019 18:24:00	8/28/2019 18:24:00	8/28/2019 18:24:00	8/28/2019 18:24:00
8/28/2019 18:25:00	No Incident	No Incident	No Incident
8/28/2019 21:03:00	8/28/2019 21:03:00	8/28/2019 21:03:00	8/28/2019 21:03:00
8/28/2019 21:04:00	8/28/2019 21:04:00	8/28/2019 21:04:00	8/28/2019 21:04:00
8/28/2019 21:05:00	8/28/2019 21:05:00	8/28/2019 21:05:00	8/28/2019 21:05:00
8/28/2019 23:10:00	8/28/2019 23:10:00	8/28/2019 23:10:00	No Incident
8/28/2019 23:11:00	8/28/2019 23:11:00	8/28/2019 23:11:00	8/28/2019 23:11:00
8/28/2019 23:12:00	8/28/2019 23:12:00	8/28/2019 23:12:00	8/28/2019 23:12:00
No Incident	No Incident	No Incident	8/29/2019 0:56:00
8/29/2019 0:57:00	8/29/2019 0:57:00	8/29/2019 0:57:00	8/29/2019 0:57:00
8/29/2019 0:58:00	8/29/2019 0:58:00	8/29/2019 0:58:00	8/29/2019 0:58:00
8/29/2019 0:59:00	8/29/2019 0:59:00	8/29/2019 0:59:00	No Incident
8/29/2019 2:06:00	8/29/2019 2:06:00	8/29/2019 2:06:00	8/29/2019 2:06:00
8/29/2019 2:07:00	8/29/2019 2:07:00	8/29/2019 2:07:00	8/29/2019 2:07:00
8/29/2019 4:10:00	8/29/2019 4:10:00	8/29/2019 4:10:00	8/29/2019 4:10:00
8/29/2019 4:11:00	8/29/2019 4:11:00	8/29/2019 4:11:00	8/29/2019 4:11:00
8/29/2019 4:12:00	8/29/2019 4:12:00	8/29/2019 4:12:00	8/29/2019 4:12:00
8/29/2019 6:58:00	8/29/2019 6:58:00	8/29/2019 6:58:00	8/29/2019 6:58:00
8/29/2019 6:59:00	8/29/2019 6:59:00	8/29/2019 6:59:00	8/29/2019 6:59:00
8/29/2019 7:00:00	8/29/2019 7:00:00	8/29/2019 7:00:00	8/29/2019 7:00:00

8/29/2019 9:38:00	No Incident	8/29/2019 9:38:00	8/29/2019 9:38:00
8/29/2019 9:39:00	8/29/2019 9:39:00	8/29/2019 9:39:00	8/29/2019 9:39:00
No Incident	8/29/2019 9:40:00	No Incident	No Incident
8/29/2019 12:16:00	No Incident	8/29/2019 12:16:00	8/29/2019 12:16:00
8/29/2019 12:17:00	8/29/2019 12:17:00	8/29/2019 12:17:00	8/29/2019 12:17:00
8/29/2019 12:18:00	8/29/2019 12:18:00	8/29/2019 12:18:00	8/29/2019 12:18:00
No Incident	8/29/2019 12:19:00	No Incident	No Incident
No Incident	8/29/2019 14:02:00	No Incident	No Incident
8/29/2019 14:03:00	8/29/2019 14:03:00	8/29/2019 14:03:00	8/29/2019 14:03:00
8/29/2019 14:04:00	No Incident	8/29/2019 14:04:00	8/29/2019 14:04:00

Table 4 indicates possible causes for the incidents in four operating metrics.

Table 4: Causes behind the discussed incident

Incidents	**Causes**
Backend Request	Batch workload
Database Latency	Lack of dynamic scaling
Backend Latency	High database latency
Frontend Latency	High database latency

Clearly in this particular case study, the bottleneck lies the database latency which is caused by the lack of dynamic scaling.

Conclusion: Modern applications need to serve different workload types with different frequencies and peak values. Some of them might vary regularly like the frontend workload in this case study, while some may vary in spikes. Handling spiky workloads in resource efficient manner is a key requirement for modern cloud service management. That is where AdeptDC's incident detection solution can help in following ways:

1. It is impossible to predict spikes in batch workloads typically coming from high volume data processing requirement for batch analytics. A supervised algorithm can help an SRE and a developer understand when the spikes are taking place.
2. It is important to understand the resource allocation landscape to handle spiky workload. Failure to do so would lead to undesirable application slowdown as shown in this case study.
3. 4 Xx error increase often happens in an unpredictable fashion. An unusual rise in 4 Xx, as detected in Figure 9 for Backend Service, does not augur well about the service health. An early detection a avoid serious future degradation.