Automatic Performance Modelling from Application Performance Management (APM) Data: An Experience Report

Paul Brebner
CTO, Performance Assurance Pty Ltd
www.performance–assurance.com.au
Canberra, Australia
paul@performance-assurance.com.au

ABSTRACT

Traditional testing approaches for enterprise systems are no longer possible, agile enough, affordable, or accurate in many cases. This is due to ongoing changes, reduced time between production updates and the inability to test all system components because of third party services and the expense of maintaining a test environment. One alternative approach has been to manually build predictive performance models to mitigate performance risk. Even this has become impractical and cannot keep pace with changes in complex enterprise systems.

In response to these challenges we have developed a way to automatically build and parameterize performance models for large scale enterprise systems from Application Performance Management (APM) data. This industry experience report summaries our experiences with automatically building performance models for commercial customers over the last two years. For each project we summarize the problem context, the performance risks to be addressed, the automatic modelling process, the range in complexity of the resulting models, the accuracy of the predictions, and the benefits and limitations of the models in practice.

Keywords

Modelling; measurement; performance; scalability; automatic; APM

1. Introduction

Since 2007 we have developed a software performance modelling tool and applied it to a variety of different enterprise technologies and performance problems [15-27]. For the last 2 years we have been delivering performance modelling in a purely commercial context which has necessitated a change from manually built performance models to automatically built models. We have found that manually built models cannot be built with sufficient speed, reliability, accuracy or complexity, to satisfy the business and technical constraints of commercial performance model delivery. However, many customers now have increasingly sophisticated APM tools installed, from a decreasing number of vendors, which provide a valuable source of data for automatic model building, particularly if the investment in building tools for conversion from APM tools to a modelling platform can be maximized by reuse for multiple customers.

Previous work in automated performance modelling explores a variety of approaches for data capture (including modelling from architecture and design artefacts, source code, code instrumentation, JVM, logs, and monitoring data), and application domains (e.g. enterprise software, HPC) [1-10].

Because we are working in a commercial enterprise application context we have focused on a few APM vendors who provide monitoring solutions which include: Large-scale monitoring of heterogeneous software stacks; End-to-end transaction flow data for every transaction and every sub-system called; a rich variety of per transaction metrics; and means of obtaining the data from the APM system (e.g. export facilities or REST APIs).

Our performance modelling tool is model driven and supports a meta-model of software performance based around SOA concepts of workloads, composite and simple services, and servers. These can be used to model business processes, workflows, and internal software processes and services to an arbitrary level of detail. Our tool provides a graphical user interface to build, parameterize and visualize performance models. The performance models are solved with a discrete event simulation engine and performance metrics computed and displayed graphically for analysis. Models can be changed and predictions easily compared. The tool is currently hosted as a SaaS to reduce costs and increase ease of use by customers. Previously we parameterized models from a variety of data sources depending on availability such as SLAs, sub-system time budgets, benchmarks, experimental in-house test-bed results, log files, custom monitoring solutions, and older APM products. Such data was often substandard in terms of quality and quantity and coverage of the target systems.

We have developed an automatic model building pipeline for one main APM vendor, and experimented with variation of it for several others. The pipeline can ingest data from files obtained from customer copies of the APM tool, or it can use REST API calls to obtain data remotely or from a local copy of the APM tool. Once the raw APM data is read it, several processing steps occur including parsing, format/structure/semantic conversion and transformation steps into our internal normalized APM data format, error checking, statistics calculations, initial internal

ICPE'16, March 12 - 18, 2016, Delft, Netherlands
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4080-9/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2851553.2851560

model building and analytical simulation (for more sophisticated validation), and finally generation of the performance model and upload to the SaaS tool where simulations can be run and changes made to the models. Depending on the amount and type of APM data available, and the purpose of modelling, different models types can be automatically generated. Typically the tradeoff is between model simplicity and complexity, modelling for capacity vs. performance predictions, and availability of detailed transactional data from the APM tools. Being able to build different model types from the same data is also valuable for research purposes and we easily investigate the pros/cons of different model types and the impact on accuracy, power of the models, and ease of use and understandability. Changes (additions, deletions, modifications, aggregation) can also be introduced at different phases of the pipeline with different pros/cons including in the APM tool before data is imported into our system, in the data itself before or during processing, in the pre-processing tool either before, during or after internal model generation, in the outputted model, or in the SaaS modelling tool.

The bulk of the paper will focus on our experiences with automatic model building for three projects. However, two aspects of modelling that are discussed in the project summaries are first introduced in more detail to provide sufficient background. These are: Model Calibration, and Model Complexity.

1.1 Model calibration

By default our performance models are parameterized with measured data (metric time distributions). They are more accurate at loads closer to the average load that the APM data was collected at, but tend to produce optimistic predictions for response times, capacity and resource requirements at significantly higher loads. For more accurate predictions models can be calibrated with load test results, however, such calibrations are dependent on the availability, correctness, and interpretability of load test data, and are context specific as they depend on the actual load and specific server resources available. Moreover, if part of the system saturates first (which is inevitable for real world systems) then the calibrations are likely to be inaccurate for the as yet unsaturated sub-systems. In theory it is be possible to obtain 0% model error using calibration data, however, in practice we have found that this is unachievable for commercial systems due to factors such as: problems running and interpreting load test result; differences in transactions and transaction mixes in load tests and models; and ambiguity around the modelling, allocation and setting resource limits for servers for detailed time breakdown metrics available from some APM tools, e.g. CPU (user/system), I/O, suspension, wait, and synchronization times. Introducing load dependent scaling (e.g. using linear or non-linear regression analysis) to model times under varying loads is a typical solution used to increase the predictive accuracy. However, it depends on having a sufficiently large sample of data from a large enough range of loads to be significant, extrapolation beyond the measured load range is risky, and it is difficult to apply to complex models which are parameterized with response time distributions, as in many cases load dependency is specific to each transaction, service and server combination. Hence, for the reported projects most of the models could not be calibrated to 0% error.

1.2 Model complexity

Model complexity is a simplified estimate of the upper number of components in a model and is computed as *complexity = (transactions * services) + services + servers*. Transactions are the number of transaction types (not individual transactions). Services appear twice as there is one model component per transaction type per service for the probability of the service being called and the time distribution, and another component per service per model representing the deployment of that service on a server. Servers is the number of physical or virtual servers the services are deployed to. Actual models may be simpler as not all services are called for each transaction, but more complex in that this formula does not take account into account relationships between components. Automatically built models can also include other system aspects such as Entry points (where a transaction enters the system), APIs, Classes and even method calls, which all have the potential for dramatically increasing the complexity but also the utility of models. Our previously manually constructed and parameterized models were practically limited to a maximum complexity of around 100, but automatically built models can have orders of magnitude higher complexity. We summarize the automatically built models for the projects below in terms of minimum (aggregated transactions) and maximum (all transaction types represented) complexities.

2. Projects

2.1 Project 1

The context of project 1 was the migration of multiple different applications from a legacy physical infrastructure that was soon to be retired, to a new virtualized infrastructure. There were a large number of poorly understood legacy applications, with different performance characteristics, and the client wanted to be able to assure their customers (who owned and used these applications) that they would still perform on the new infrastructure well in advance of migration. They also wanted to know if the sizing of the new infrastructure was sufficient and also efficient from a cost perspective. The plan was to take an example application (the first application to be migrated), install an APM tool on the physical infrastructure, deploy the application on both platforms, and run load tests on each application and platform to determine the maximum capacity. The next step was to build models from the APM data on the physical platform, and then demonstrate that given the APM data and model for the application on the physical platform, the performance, capacity and resource requirements for the application on the new virtualized platform could be predicted, and that this was a repeatable process for other applications to be migrated in the future.

Our methodology was to run a load test multiple times for each application/platform, and ramp up the load gradually until maximum throughput was observed. Load test and APM data was captured for each test and the results used to build a model and validate the predictions. The high level results are as follows. For the old physical platform, the maximum measured capacity was 14TPS. Models were built for the lowest and the highest loads. The lowest and highest load models predicted an optimistic maximum theoretical capacity of 92TPS and 63TPS respectively, both obviously higher than the actual measured capacity.

For the new virtualized platform, the maximum measured capacity was higher at 38TPS. The low load model from the physical platform was calibrated for the lowest and the highest loads on the virtualized platform, and predicted a closer maximum theoretical capacity of 45TPS and 49TPS respectively, with an error of 21% and 33% respectively.

The uncertainty in these predictions is due to a combination of factors including using only load independent models which don't take into account increasing times with increasing load, uncertainty over how to allocate and accurately model capacity of time metric breakdowns (User and system CPU/IO, and other times), and simplifications made in calibrating the models (using scaling factors computed using average times for each server at the maximum load divided by the average times for the same components at the minimum load. To be more accurate, calibration is ideally done per transaction type/service).

Another metric was also computed which we call "efficiency". It is computed as *efficiency = TP/RT/cores*. Higher efficiency is better. The old physical platform was actually faster than the new virtualized platform, although the new virtualized platform had higher concurrency (16 cores c.f. 12). The efficiency of the old physical platform was better (7.8) than the efficiency of the new virtualized platform (5.8). The metric *TP/RT* is referred to in the literature as "power" [11, 12].

The business goal of this project was to reuse the methodology to assist with migrating multiple applications. However, there was a problem with repeatability with our approach, as in theory the calibration factors used to scale the model of the application on the legacy platform to make accurate capacity predictions for the new platform first needs to be obtained from load tests of the application on the new virtualized platform – but the application has to be first deployed on the new platform. The calibration factors are likely to be specific to application, hardware, virtualization technology, cores per VM, load, etc. However, a few more complete iterations with multiple application examples would produce more calibration examples. It may then be possible to calibrate models with a range and average values obtained from previous examples, to provide sufficient assurance before decisions are made to migrate the applications to the new virtualized platform.

The project 1 application was relatively simple, having 40 transaction types, 2-3 services, and 3-4 servers. The model complexity for the application on the physical platform was 85 and for the virtualized platform the model complexity was 127. We used 24k transactions from the APM tool to build the model on the physical platform, and 67k on the virtualized platform. On the physical platform the average metric breakdown percentages were 44% for DB time, 2% user CPU, and 53% system CPU/IO. On the virtualized platform the average metric percentages were 11% for DB time, 1% user CPU, 87% system CPU/IO. In absolute terms both DB time and system CPU/IO times were greater on the virtualized platform, probably due to the overhead of virtualization.

2.2 Project 2

The context of project 2 was the migration of a large scale mission critical application to a web-based system, and migration to new infrastructure to support it. There was a hard deadline for the migration to occur by, and the client wanted assistance in assuring that the migrated application had adequate performance and scalability, and that the new infrastructure was sufficient but not substantially over resourced, as there was a significant cost overhead and time delay for server provisioning. The development and testing and user acceptance testing of the new system was conducted on different platforms to the new production infrastructure, and the focus of testing was functional rather than performance or load testing. However, all the systems were monitored by a commercial APM tool which provided detailed per transaction metrics per service/sub-system, including a breakdown of metric times (user and system CPU, I/O, wait, synchronization and suspension) and server metrics including CPU utilization and number of cores. Due to the timetable of the migration and when data from the various testing activities was available, we had a very limited amount of time to obtain the APM data, build models, and make capacity and resource predictions.

The earliest APM data available was from functional testing in an Amazon Web Services (AWS) Cloud environment. The tests were performed in a planned order, exercising a subset of transaction types at a time over a period of weeks. We therefore needed to obtain the APM data for a significant range of the test period to obtain as many transaction types as possible. This gave us 104k transaction samples to work with. Once we had the data we analyzed it to check for quality prior to modelling, revealing that there were a large number of errors and unhandled exceptions, and the majority of time was spent in synchronization and I/O+System CPU (50% synchronization, 25% I/O+System CPU, 20% User CPU). There were 2114 transaction types in the test data, significantly less than expected, leading us to suspect that the data was not giving us complete coverage of the application. It was also unlikely that the transaction ratios from test were representative of a production mixture. After filtering out some of the worst transactions, we automatically built and parameterized a performance model from the remaining data. The most complex model that we built from this data (including all 2214 transaction types, 27 services, and 46 hosts) had a complexity of 59k, and the simplest (using a single aggregated transaction, which is adequate for capacity prediction) had a complexity of only 100. Using only the CPU time breakdowns the model predicted a total minimum number of cores at the target load of 240.

Given the potentially problematic results from the AWS environment, the client decided to accelerate production deployment, and deployed a subset of the final functionality onto a smaller production environment earlier than planned. We obtained 80k transactions over 5 days of APM data from this environment. We noticed that the percentage of User CPU time had gone up to 50% (but absolutely was a lot less), I/O+system CPU time was 20%, synchronization time had dropped below 1%). There was also a substantial reduction in errors and unhandled exceptions. The number of transactions types had increased to 7771 so more functionality was being measured. The model automatically built from this data predicted a total minimum number of cores at the target load of 74, significantly less than the original prediction. The client had meanwhile arranged for a cheaper/more flexible provisioning option and proceeded to fully deploy the application in production with 152 cores (approximately double the minimum predicted cores) to reduce the risk of performance and scalability issues. Once the production system had been deployed and running for several weeks we obtained a final 220k transaction sample. The final number of transaction types was 13k, with 30 servers and 46 servers. The most complex model had 402k components, but the simplest had 106 components. The final model predicted a minimum of 72 cores at the target load, close to the 2nd prediction. However, we noticed that the percentage of user CPU time had dropped to 13%, synchronization and suspension times had risen to 3%, but I/O+system CPU had risen to 74%. Given that the application was consuming significant non-user CPU resources, and that as the load increased some of these times would increase due to load dependence, we redid the prediction including these times, giving a more pessimistic minimum number of cores at the target load of 134, closer to the number the client had used in production (15% error).

This project exposed some difficulties of building performance models from APM data which was skewed in multiple ways (e.g. exceptions, incomplete functionality, un-representative transaction ratios, from different infrastructures, at low load compared to the target load). The main challenge was calibrating the model for accurate capacity predictions at significantly higher target loads than measured. The average measured load from the three samples of APM data was 12k, 8k and 30 times less than the target load, so the potential calibration error was substantial. Even though it was possible to rapidly build performance models to assist with capacity prediction, project 2 suggests that having high quality representative APM data is desirable for improved accuracy of results. In practice it is also becoming common practice to have a more flexible and dynamic provisioning approach (in terms of costs, initial and ongoing billing periods, VM sizes, provisioning times, elastic spin up of VMs, etc), and this will better manage unknowns, variations, and spikes in loads.

Note that as this project was focused on capacity prediction the simplest possible models (a single aggregated transaction) were used. The more complex models mentioned were built as a proof of concept to show that it was possible to build models that could also be useful for predicting response times for each transaction type.

2.3 Project 3

The business goals for project 3 were to demonstrate that we could build performance models automatically from a proprietary APM solution. The models had to be accurate for the baseline systems, and we had to demonstrate that we could model a number of alternatives of relevance to the customer. The system was a mission critical distributed system with many types of online users (employees and users, in different locations, on different platforms), it had well defined SLAs for the internal processing time for the system (excluding network time), and performed risk assessment processing in real-time. Risk assessment was performed by a sub-system which supported multiple types of assessment services, consuming different amounts of resources, and called with a large range of different frequencies per day. The challenges for the client were: to support a growing number of users over time; different peaks per day depending on the time of year; to develop, test, and put into production increasing numbers of risk assessment services at a faster rate than before; to explore alternative deployment options, all while understanding the impact of changes on performance, scalability and resources.

We have previously had experience of automatically building performance models from an APM product which provided detailed application independent per transaction metrics in a variety of data formats including XML and CSV, for which we developed a pre-processor (Projects 1 and 2). However, the semantics, structure and format of the data from this clients' APM tool were different. We developed a solution in Apache Hive [28] to pre-process the client APM data and transform it into the semantics, structure and format that could be consumed by our automatic model building pipeline. The main challenge was that the times from the APM tool were provided for "profile points" (times which were captured by developers in their code using the APM tool harness, basically a way of registering the time between two arbitrary points in the code). These times were developed for assurance, testing and business purposes rather than performance modelling, and therefore sometimes crossed sub-system and server boundaries, requiring extra processing to determine the breakdown of times for each sub-system and server.

Once the new enhanced APM modelling pipeline was developed and tested we were able to repeatably build baseline models from a day or more of production or test data. The baseline model response time performance predictions were accurate to less than 10% error for the baseline average daily load giving us and the client confidence that modelling alternatives would be useful. We have a number of different ways of modelling alternatives with this type of automatic model building pipeline including: deleting, changing or adding to the input APM performance data (e.g. by increasing the number of some type of transactions, by introducing new transactions and times, etc); by introducing transformations into the model building phase (e.g. categorizing or aggregating transactions differently, scaling times, or changing the service to server deployment mappings, or by changing the number or CPU on servers); or by manually or semi-automatically changing the resulting models. For this project all of these approaches were trialed. Some simple alternative models produced included: Calling the risk-assessment services either synchronously or asynchronously, calling some sub-systems sequentially or concurrently (if allowed by the business logic), adding new risk-assessment services (given the expected frequency of calls per day and average response time).

A more complex alternative scenario was to model the impact of splitting the risk assessment services across multiple different servers. The default deployment was for each service to be deployed on multiple servers, but the number and type of services was close to the upper limit that could be supported with the existing infrastructure due mainly to RAM consumption. The client was interested in modelling the impact of deploying a subset of the services on different numbers of servers. This is actually an optimization problem, comparable to the box packing problem with multiple variables (number of services, number of servers, CPUs per server, memory use per service, response time distribution per service, and frequency of calls per service - average and peak). The client had determined a possible split of services to servers, but it was apparent from the initial base model that we had built that some services were substantially more demanding than others (in terms of Service Demand), which they had not taken into account.

As an aside, we recently noticed [personal email to Samuel Kounev and Andreas Brunnert, 1/9/2015] that (based on upwards of five client examples of APM data) the service demand of components, sub-systems, services, etc of large scale distributed systems typically approximates Zipf's law. That is, a few components are more demanding that all the others put together. A graph of log10 of *rank* of service demand for each component vs. log10 of service demand for each component gives a straight line with a gradient close to -1.0. Zipf's law has been explored for word frequency, city sizes, animal species, the internet, and performance evaluation, etc [13, 14]. This relationship has potentially interesting/useful implications for software performance and modelling. For example, the scalability and resource usage of a system may be critically dependent on the very few most demanding components. Also, if you can measure the demand on the most demanding component, and estimate the total number of components, Zipf's law can be used to estimate the total service demand of the system (e.g. in the same way that given the weight of just the largest animal and the number of species on Noah's Ark, you can estimate the total mass of animals on the Ark).

We modelled the impact of locating only the most demanding four risk assessment services on one server, and all others (approximately thirty) on another server. This resulted in a predicted 50:50 resource split so was optimal in terms of resource load balancing. Even though it doesn't measure the impact for all the variables, it does demonstrate the possibility of significant problems with performance and scalability if the wrong subsets of assessment services are deployed on the same server. For example, scalability and therefore response times under higher loads will be impacted if too many high demand services are deployed to one server only. Response times (potentially even at lower loads) may be impacted if services with very long and very short response times are deployed on the same server. We have therefore proposed a future work plan to the client to address this risk. They will provide us with: details of service performance metrics (including number, memory use, response times, and frequency of use), and possible deployment scenarios. We will optimize the deployment of the services for 1-n servers with different numbers of CPUs, and model the most promising deployments and predict response time distributions and resource usage/scalability and tell them which ones pass/fail SLAs.

There were a couple of challenges to producing an accurate capacity model for this project. The 1st was that the APM tool did not give us a breakdown of time metrics, just response time. If breakdown times are available we have previously found that it is possible to calibrate models for best and worst case capacity predictions, and at least for simple (e.g. linear) load dependence. However, the APM data did give us explicit Garbage Collection times (because GC was one of the profile points), which revealed that GC was a significant overhead for the system (33% of the total service demand). Consequently a simple tactic to improve resource usage was to explore more efficient GC strategies, such as the use of multiple object pools to enable objects used in only a single transactional context to be efficiently deleted.

The 2nd problem related to the limited range of load data. The baseline model was built from a typical day's production data. The peak load of the day was close to the expected absolute peak load for the application, there was not much variation in load across the day, and there was still substantial headroom available on the servers. It was therefore difficult to calibrate the model for load dependent behavior to accurately predict response times and resource demands at significantly higher loads using techniques such as regression. Instead we relied on the client conducting a stress to break load test. However, the test itself had problems with performance and resource problems (increased response time, increased CPU utilization, increased errors) well before the final eventual stable measured maximum capacity was reached. Nevertheless, using the final measured maximum throughput we calibrated our baseline model to predict the maximum throughput accurately, resulting in a reduction in error margin from 60% to 10%.

One of the unexpected difficulties with the APM data for this project was that for some likely modelling alternatives a model will be needed which accurately captures the different transaction types. We have successfully built detailed transaction type models from other APM data, however it was unavailable for this application and APM combination. Instead we attempted to infer the transaction types from the available APM data, using a combination of parameter values visible in the data and services called. We were able to automatically produce multiple possible transaction categories, however producing the exact required category was unachievable as in practice the resulting categories were either too few or too many.

The baseline model was finally built from several days of data (170k transactions), with half of the data used to build the model, and half to validate it. The simplest model had 1 transaction type, 44 services and 7 servers, giving a model complexity of 95, and the most complex had 183 inferred transaction types giving a much higher model complexity of 8k. The initial APM data size was 2GB, so the size of each transaction was 11.8KB on average. The output from Hive pre-processing was 34MB, and the model size (XML size before upload to our SaaS modelling tool) was 235KB, giving an overall size reduction of 8,510 times. This shows quantitatively the power of abstraction due to modelling, even simply at the level of the data compression ratio from raw APM data to a model.

As part of proposal for providing ongoing modelling services for this client we estimated the likely ROI of using a hybrid testing and modelling approach. We assumed that each load test takes 1 week and costs $10,000, and that the initial performance model also takes 1 week and costs $10,000. Evaluating alternatives using the performance model are much faster and cheaper, taking only 1 day and costing $2,000. A testing only approach can only complete 4 tests per month at a cost of $40,000. A hybrid testing/modelling approach, allowing for 1 initial model build, 4 alternatives modelled and evaluated, and 1 load test to confirm the most promising alternative, would cost $28,000 and take 2 weeks and 4 days. Better cost and time savings can be achieved as the effort to build initial and updated models decreases over time (potentially reducing to a single day), and with increasing numbers of alternatives modelled. This increased efficiency is likely to have agility benefits as decisions can be made earlier based on modelling results, even in the business and development phases.

3. Conclusions

Table 1 shows a summary of the projects including minimum and maximum model complexity, and the smallest error % for each.

Table 1 Summary of model complexity and smallest error

	Minimum Complexity	Maximum Complexity	Smallest Error %
Manual models	3	100	<= 100%
Project 1	85	127	21%
Project 2	100	60k	15%
Project 3	95	8k	10%

Some issues we discovered include: lack of standards for APM data resulting in differences in metrics available, different metric semantics and names, differences in metric structure and availability of transactional metrics. Accurate and automatic calibration of models for load dependency and maximum capacity is an outstanding problem. Some of the models quickly become very complex, making it difficult to visualize them and make manual changes. The benefits of automatic model building are significant. We demonstrated that we can build models automatically for a variety of application types and from several APM tools. The use of detailed transaction metrics including response time distributions rather than just averages results in models with accurate response time distribution predictions. Models are useful, accurate enough in practice, and can be built fast and reliably enough to solve real business problems on a commercial basis.

3.1 Future work

We discovered that some APM tools and applications capture a very large amount of data per transaction. Consequently one of the significant issues we encountered was the long time taken to obtain data for large numbers of transactions from the APM tool. In the worst case we could only extract 10 transactions per second from the APM REST API. We did not need all the data captured about each transaction to build performance models, but there was no way to request only the required data, which may have increased the throughput. However, once the data was obtained from the APM tool our automatic model building program worked very quickly (in the order of seconds, rather than hours to get the data in the first place). Apart from obvious potential improvements to the APM APIs to improve throughput, another solution we have considered is sampling. For some APMs it may be possible to obtain summary data for all the transactions of interest very quickly (including transaction type, start and end time of transaction, total response time, and transaction id). In this case it is often possible to obtain detailed transaction data for specific transactions by request. Some sampling strategies we have either experimented with or are considering for the future include: model building from a small fixed or percentage random sample of transactions (using complete data), and testing against the remainder (using summary data only); incrementally building models starting from very small sizes and then increasingly the sample size until there is no observed benefit from including more samples; filtering transactions to remove problematic samples with errors or exceptions; filtering transactions based on load ranges or thresholds, e.g. to remove samples with long response times, or when server or thread resources are saturated; and requesting samples of specific transaction types to ensure that each transaction type has sufficient representation to be statistically significant. Note that some APM tools provide ways of filtering the transactions in advance of (or as part of) the external service calls to obtain the data.

Some of these approaches will also be applicable for use in DevOps when there is a continuous stream of APM data available. Models could be built periodically, incrementally or even continuously. Full or incremental or continuous model building, or partial updates to models, could be triggered by changes to components in the monitored environment (e.g. transaction types, services or servers coming or going) and once significant differences between model predictions and APM measurements are detected. Models could be quickly rebuilt once a potential problem is detected using the most recent data only, and then used to compare with previous model results.

We are also investigating the use of automatic decisions about possible model types based on analysis of the APM data and model purpose. This could also result in automatic retrieval of more, better or selected data on demand to achieve modelling purposes with the least effort and most agility.

We also suggest that it would be valuable to the wider performance engineering community to conduct more in-depth experiments and comparisons of different APM tools on the same application, with the purpose of evaluating what types and accuracy of models can be built from each, what data is essential or optional, and overall what the pros and cons of each APM tool are for automatic model building. To facilitate this we have developed a draft evaluation framework based on knowledge of four APM tools which begins to capture important similarities and differences.

4. REFERENCES

[1] Torsten Hoefler, "Towards fully automated interpretable performance models", PASAC16, June 2016, http://icl.cs.utk.edu/newsletter/presentations/2015/Hoefler-Towards-Fully-Automated-Interpretable-Performance-Models-2015-07-01.pdf

[2] Michael Hauck, "Automated Experiments for Deriving Performance-relevant Properties of Software Execution Environments", KIT Scientific Publishing, February 2014.

[3] A Mizan, G Franks, "Automated Performance Model Construction through Event Log Analysis", ICST2012, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6200164

[4] Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. 2009. Automated anomaly detection and performance modeling of enterprise applications. *ACM Trans. Comput. Syst.* 27, 3, Article 6 (November 2009), 32 pages. DOI=10.1145/1629087.1629089 http://doi.acm.org/10.1145/1629087.1629089

[5] Andreas Brunnert, Christian Vögele, Helmut Krcmar: "Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications", EPEW 2013: 74-88

[6] Felix Willnecker, Andreas Brunnert, Wolfgang Gottesheim, and Helmut Krcmar. 2015. Using Dynatrace Monitoring Data for Generating Performance Models of Java EE Applications. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering* (ICPE '15). ACM, New York, NY, USA, 103-104.

DOI=10.1145/2668930.2688061 http://doi.acm.org/10.1145/2668930.2688061

[7] Markus Dlugi, Andreas Brunnert, and Helmut Krcmar. 2015. Model-based performance evaluations in continuous delivery pipelines. In *Proceedings of the 1st International Workshop on Quality-Aware DevOps* (QUDOS 2015). ACM, New York, NY, USA, 25-26. DOI=10.1145/2804371.2804376 http://doi.acm.org/10.1145/2804371.2804376

[8] Wu, Xingfu, Valerie Taylor, and Joseph Paris. "A web-based prophesy automated performance modeling system." *the International Conference on Web Technologies, Applications and Services (WTAS2006)*. 2006.

[9] Tauseef A. Israr, Danny H. Lau, Greg Franks, and Murray Woodside. 2005. Automatic generation of layered queuing software performance models from commonly available traces. In *Proceedings of the 5th international workshop on Software and performance* (WOSP '05). ACM, New York, NY, USA, 147-158. DOI=http://doi.acm.org/10.1145/1071021.1071037

[10] Murray Woodside, "Performance Data and Performance Models", Keynote, First SPEC Int. Performance Evaluation Workshop, 2008, http://www.sipew2008.org/presentations/SIPEW08-Keynote-Woodside.pdf

[11] Dror G. Feitelson and Larry Rudolph. 1998. Metrics and Benchmarking for Parallel Job Scheduling. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* (IPPS/SPDP '98), Dror G. Feitelson and Larry Rudolph (Eds.). Springer-Verlag, London, UK, UK, 1-24. http://www.cs.huji.ac.il/~feit/parsched/jsspp98/p-98-1.pdf

[12] L. Kleinrock, "Power and deterministic rules of thumb for probabilistic problems in computer communications". In Intl. Conf. Communications, vol. 3, pp. 43.1.1-43.1.10, Jun 1979.

[13] Zipf's law, references: http://ccl.pku.edu.cn/doubtfire/NLP/Statistical_Approach/Zip_law/references%20on%20zipf's%20law.htm

[14] Mark Crovella. 2000. Performance Evaluation with Heavy Tailed Distributions. In *Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools* (TOOLS '00), Boudewijn R. Haverkort, Henrik C. Bohnenkamp, and Connie U. Smith (Eds.). Springer-Verlag, London, UK, UK, 1-9.

[15] Brebner, P. C. 2012. Experiences with early life-cycle performance modeling for architecture assessment. In *Proceedings of the 8th international ACM SIGSOFT Conference on Quality of Software Architectures* (Bertinoro, Italy, June 25 - 28, 2012). QoSA '12. ACM, New York, NY, 149-154. DOI= http://doi.acm.org/10.1145/2304696.2304721

[16] Brebner, P. C. 2012. A performance modeling "blending" approach for early life-cycle risk mitigation. In *Proceedings of the 3rd ACM/SPEC international Conference on Performance Engineering* (Boston, Massachusetts, USA, April 22 - 25, 2012). ICPE '12. ACM, New York, NY, 271-274. DOI= http://doi.acm.org/10.1145/2188286.2188336

[17] Brebner, P. C. 2012. Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (IaaS) cloud applications. In *Proceedings of the 3rd ACM/SPEC international Conference on Performance Engineering* (Boston, Massachusetts, USA, April 22 - 25, 2012). ICPE '12. ACM, New York, NY, 263-266. DOI= http://doi.acm.org/10.1145/2188286.2188334

[18] Brebner, P. C. 2011. Real-world performance modelling of enterprise service oriented architectures: delivering business value with complexity and constraints (abstracts only). *SIGMETRICS Perform. Eval. Rev.* 39, 3 (Dec. 2011), 12-12. DOI= http://doi.acm.org/10.1145/2160803.2160813

[19] Brebner, P. C. 2011. Real-world performance modelling of enterprise service oriented architectures: delivering business value with complexity and constraints. In *Proceedings of the 2nd ACM/SPEC international Conference on Performance Engineering* (Karlsruhe, Germany, March 14 - 16, 2011). ICPE '11. ACM, New York, NY, 85-96. DOI= http://doi.acm.org/10.1145/1958746.1958762

[20] Brebner, P. and Liu, A. 2011. Performance and cost assessment of cloud services. In *Proceedings of the 2010 international Conference on Service-Oriented Computing* (San Francisco, CA, December 07 - 10, 2010). Springer-Verlag, Berlin, Heidelberg, 39-50.

[21] Paul Brebner, Is your Cloud Elastic Enough? Part 1. CMG Measure IT, Issue 2, 2011. http://www.cmg.org/wp-content/uploads/2011/08/m_82_3.pdf

[22] Paul Brebner, Is your Cloud Elastic Enough? Part 2. CMG Measure IT, Issue 3, 2011. http://www.cmg.org/wp-content/uploads/2011/10/m_84_3.pdf

[23] Brebner, P. 2009. Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application. In *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications* (August 27 - 29, 2009). IEEE Computer Society, Washington, DC, 404-411. DOI= http://dx.doi.org/10.1109/SEAA.2009.57

[24] Paul Brebner, Liam O'Brien, Jon Gray: "Performance modeling evolving Enterprise Service Oriented Architectures". In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. 14-17 Sept. 2009. 71 – 80. DOI=http://dx.doi.org/10.1109/WICSA.2009.5290793

[25] Brebner, P., O'Brien, L, Gray, J., "Performance modeling power consumption and carbon emissions for Server Virtualization of Service Oriented Architectures (SOAs)". EDOCW 2009. 13th. 92-99.

[26] Paul Brebner, Liam O'Brien, Jon Gray. "Performance Modeling for e-Government Service Oriented Architectures (SOAs)", ASWEC (Australasian Software Engineering Conference) Proceedings (Perth, March, 2008), 130-138.

[27] Paul C. Brebner. 2008. Performance modeling for service oriented architectures. In *Companion of the 30th international conference on Software engineering* (ICSE Companion '08). ACM, New York, NY, USA, 953-954. DOI=http://dx.doi.org/10.1145/1370175.1370204

[28] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - A Petabyte Scale Data Warehouse Using Hadoop. In ICDE, 2010.