Triangulating Performance Bottlenecks in Business Critical Application's

Sameer S Paradkar, Enterprise Architect

ATOS, Architecture Group, Business & Platforms Solutions Global Delivery Centre-India

Abstract **- The health of modern organizations depends on many complex business and operational processes that are impossible to monitor and manage manually. A disruption to these systems can cost millions of dollars to enterprises and jeopardize customer satisfaction and loyalty. Until recently, few analytical tools could scan through humungus data, detect subtle changes, and proactively notify users about issues that might adversely affect business outcomes. They continuously detect anomalies, trends, and correlations and present individuals with a handful of most relevant insights. However, only monitoring tools are not adequate to analyse and detect the root causes in terms of a performance issue. This paper proposes an approach that is based on 3 core pillars that are basis analysis of business systems, static analysis, and dynamic analysis.**

Modern applications carry a heavy burden: they are required to deliver high performance and are featured rich. They must provide personalization to users, assemble information from disparate and data sources, and process data intelligently depending on user requests. These applications are fragmented into multiple operations executed for the same request, the results of which are assembled into a structured response to the user. For example, the request to view an account balance may require checking for a valid session, querying a backend database, and finally putting together an HTML response leveraging the results from the database query. As some of these operations are time-consuming, such dynamic applications have high inherent costs in performance and scalability. This paper will also discuss the different technique for building high performance business applications.

Index Terms **- Performance Bottlenecks, Static Analysis, Dynamic Analysis, Hotspots, Blind Spots, Vulnerabilities, Architecture Trade Analysis, Application Performance Management**

INTRODUCTION

In every application, performance matters the most. Research has proved that if an application takes more than 2 seconds to respond to end-user requests end-used tend to leave or quite the application. When end-Users complain about performance and unresponsive features, many of these factors lead to losing those users as customers. To satisfy users' needs, performance must be continuously checked, profiled, and monitored. Performance profiling & optimization occurs by monitoring and analyzing the performance of the application and identifying ways to improve it. An application are a mixture of server-side and client-side components. The application can have performance problems on either side, and both need to be profiled and instrumented. The client side relates to performance as seen within the web browser. This includes initial page load time, downloading resources, JavaScript that runs in the browser etc. The server side relates to how long it takes to run on the server to execute requests. One of the ways optimizing the performance on the server generally revolves profiling & optimizing things like database queries and other application dependencies.

The application delivery chain will continue to grow as businesses continue to leverage newer technologies such as virtualization, cloud services, or mobile enterprise apps to help drive better efficiency or competitive differentiation. Applications have changed and have become "composite," meaning that multiple elements are assembled for the first time within the end user's browser. For instance, some services or elements come from within your data center, some through CDNs and others from third parties delivering ratings and reviews, ads, news feeds, e-commerce, and more. In fact, the average application consists of 10 separate services — each of which impacts end user experience, availability and response time. And while IT monitors pieces of the application delivery chain, they lack the complete picture of how performance is delivered to their users.

Performance Bottlenecks in Business-Critical Applications – Problem Context

The traditional approach is considered bottom-up and does not provide full visibility into the entire business transaction between the end user and back-end services. Individual IT support teams using disparate APM toolsets do not work collaboratively, creating inefficiency and longer mean time to repair (MTTR) to support business needs and IT objectives.

Enterprise application landscapes consists of services and functionality that is delivered to the end users from multiple sources. These sources include components delivered from the data center as well as components delivered from third parties and the cloud, such as content delivery networks (CDNs), news feeds, ads, analytics, bill payment, and e-commerce platforms. With so many moving parts in the data center, plus virtualization, the cloud layers of application tiers, and the staggering amount of data handled by a heterogeneous infrastructure, the functional complexity is enormous.

The application value chain is more complex because of:

- Increased complexity in the data center due to multitier architectures, virtualization, web services and APIs, mobile and WAN optimization.
- Increased complexity on the web is due to third-party providers, CDNs, multiple browsers (Chrome, Safari, Explorer, etc.), and mobile devices (smartphones, iPads, etc.); the use of cloud providers.
- Saturated peering points must serve a mix of third-party providers, CDNs, cloud providers, multiple browsers, and mobile devices — leading to uneven performance. Moreover, if the last mile — the connection between an end user and an ISP — is slow for any reason, the end user experience suffers, and so might the business.

When performance problems do occur for the end user, the inability to monitor across the entire application delivery chain hinders the organization's ability to rapidly determine the route cause of the problem. And much time is spent trying to determine where the performance problem resides

Application delivery chains will continue to grow in complexity. Prior to the cloud onslaught, IT managers had a much easier time troubleshooting and optimizing application performance because all underlying elements were within a single data center. In the new world of cloud service providers (CSPs) and composite applications, troubleshooting application issues requires IT to chase down multiple complex request paths for an ever-growing number of applications.

Lack of end-user performance insight and the inability to rapidly isolate fault domains across the application delivery chain leads to delays and high costs to resolve application performance problems. Developers' time is spent majorly on debugging applications and trying to reproduce problems. A large number of production problems could be avoided by implementing a more proactive approach to application performance management.

More importantly, IT does not always have insight into the end-user impact of the IT infrastructure's performance. This issue is complicated by the fact that a growing number of organizations are outsourcing critical elements of their application delivery chain to CSPs, such as CDNs, outsourced data centers, and e-commerce platforms. In such cases, there is often less insight into performance issues from the beginning to end.

Challenges faced by Different Stakeholders	
End Users	Operations Team
- Application is slow during certain periods of the day or week - Serious Application Errors that corrupt data - Improper responses from L1 Support the reason for the problem - Previous problems recurring and hampering users' ability to use the application - Low end user morale and loss of productivity	- Lack of appropriate tools to troubleshoot downtime issues - Non-integrated tools make troubleshooting time consuming false alerts - Inability to understand the real impact of downtime on the application - Shifting of responsibility among various technology teams - like network team, server team, application server administration team, database team, etc. - Lack of Actionable Data even with so many monitoring tools - High time to identify the root cause of a problem because of the large number of resources monitored - Lack of skilled resources who understand various technologies

Trangulating HotSpots, BlindSpots, and Vulnerabilities

The approach is to identify the impediments in the application's performance to find the root cause of the issues. By identifying the impediments, one can plan and select the best possible solution and an alternative to resolving the performance issues. The approach outlined in this paper is based on 3 core pillars which are asis state analysis, static analysis and dynamic analysis. As the system code base is typically very very large, one must rely on automated tools and utilities for analysis to triangulate the performance bottlenecks and hotspots. Please find the diagram that depicts the approach that the paper proposes to leverage to triangulate hotspots, blind spots, and vulnerabilities.

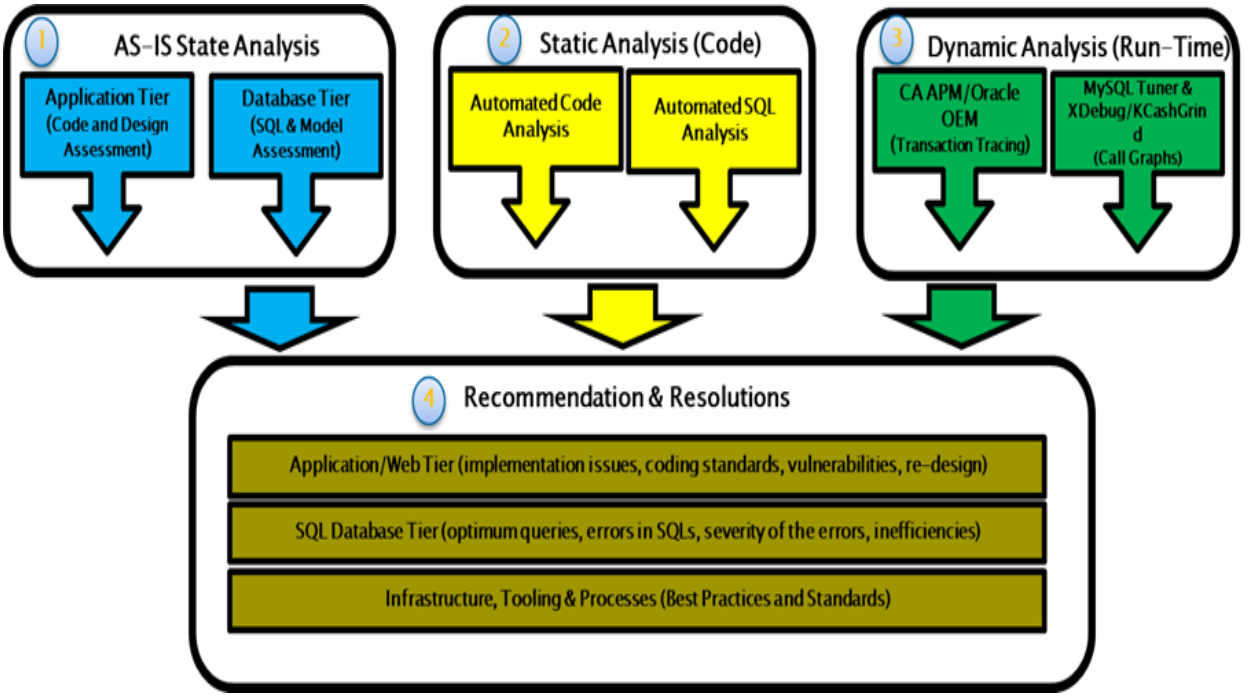

Figure: Key Pillars of Performance Assessment

Analysis the As-Is State

The principal benefit of the Architecture Analysis is the ability to see if the proposed overall software structure will live up to the user expectations. The architecture analysis helps one understand the strengths and weaknesses of a system. The method not only helps evaluate the current system architecture, but also helps with the design of the architecture for a new system. This methodology helps designers to ask the right questions and solve critical issues early in the project. Additionally, selecting the right degree of quality for a specific system fully utilizes the money spent by the customer, as the money spent in any one area of the system will be justified. The analysis consists of the following aspects

Steps	Description
Architecture Analysis – Design Patterns and aligning with best practices	Analysis of the architecture type and deviations of any to determine any negative performance impact
Refactoring or removing an Anti-Patterns	Identification of Performance Anti-Patterns as their use produces negative consquenses
Alternative Component Integrations or Interactions	may be possible to change the interaction to improve responsiveness or throughput
Strategic Improvements	Other opportunities to significantly improve performance by applying Performance Principles and best practices
Tactical Improvements	Other Performance Patterns could also be applied to immediately improve system throughput

The methods described in this section are explained in the form of techniques in the next section.

Techniques for building highly performant business systems

Tier / Domain	Techniques
Architecture & Design	The efficiency and effectiveness of a performance strategy are closely tied to a caching strategy. From a performance standpoint, having an efficient and elaborate caching mechanism is the first step in the process. Take the inventory of all key application components and come up with a strategy to cache the data that will speed up the overall process.
Architecture & Design	Evaluate all possibilities of failure and their likely probability. A few common failure events could be hardware failure, security breaches, natural disasters, a sudden spike in traffic, network failure, and so on. For each of these events, attach a weight and the probability of its occurrence. Then devise a fault-tolerance mechanism for each of these events. The fault handling procedure and failover options minimize the latency issues caused by failed components.
Architecture & Design	Design solutions so that they can be distributed across multiple nodes. This offers a dual advantage of both performance and scalability.
Architecture & Design	The key components should be kept lightweight by minimizing their overall size and minimizing server round trips. The most popular way of implementing a lightweight design is to use asynchronous JavaScript and XML - AJAX-based components with no or minimum JavaScripting. This reduces the impact on page size as well as the number of page refreshes and server round-trips.
Architecture & Design	Leveraging open standards not only allows for future painless extension of the technology stack, but it also helps in understanding the technology and troubleshooting in the case of performance issues. Software components should be loosely coupled so that the failure of a performance issue on one component does not impact the overall response time.
Architecture & Design	The various tiers should be hosted on different hardware machines. Resource intensive operations and the database are deployed on dedicated hardware, providing improved performance.
Architecture & Design	Leverage resource pooling in the application container to improve the performance. Use of connection pooling for database connections will improve performance.
Architecture & Design	Load balancing is a key technique to spread the load evenly between various nodes. Load balancing or distribution through the Round Robin DNS algorithm will facilitate in superior performance.
Architecture & Design	Separate long-running critical processes that might fail by using a separate physical cluster. For example, a web server provides superior processing capacity and memory, but may not have robust storage that can be swapped rapidly in the event of a hardware failure.
Architecture & Design	The key scalability patterns include distributed computing, parallel computing, SOA, event-driven architecture, push-and-pull data modeling, optimal load sharing, enterprise portals, and message modeling.
Architecture & Design	Establish smart caching to cache the frequently used data/query results in the application tier to avoid costly APIs.
Integration & API	Lower traffic on the wire by sending only what is required and retrieving only what is necessary.
Integration & API	Reduce the number of transitions between boundaries, and reduce the amount of data transferred over the wire. Choose batch mode to minimize calls over the network channel.
Integration & API	Where communication tier boundaries are crossed, we leverage coarse-grained interfaces require a reduced number of calls for a specific process, and consider using an asynchronous model of communication.
Integration & API	Design effective locking, transaction queuing, and threading mechanisms. Leverage optimum queries for superior performance, and avoid bulk data fetching when only a subset is required for the operation. Leverage asynchronous APIs, message queuing, or one-way calls to minimize blocking while making calls across tier boundaries.
Integration & API	Architect an efficient communication methodology and protocols between tiers to ensure entities securely interact with no performance degradation.
	Largely granular interfaces require multiple invocation to perform a task and are the best solution alternatives when located on the same physical node.
Integration & API	Interfaces that make only one call to accomplish each task provide outstanding performance when the components are distributed across physical boundaries.
Integration & API	Adopt REST-based integration rather than heavyweight alternatives, such as Simple Object Access Protocol (SOAP) or Application Programming Interface (API) calls. Using lightweight alternatives, such as REST-based services, are fast, they transfer less data, and are more scalable.
Integration & API	Leverage lightweight alternatives, such as JSON over XML for service invocation, and do the service invocations only when needed.
Integration & API	Avoid chatty APIs and batch requests to minimize server round-trips. The fewer the calls, the less the load on the server and hence it will be more scalable.
UI Tier	Minimize the number of static assets, such as images, JavaScript, Cascading Style Sheets (CSS), required by the application. This can be achieved by compressing and merging them to form a minimal set.

UI Tier	Leverage the AJAX-based approach, whether it may be for client-side components communicating with the server, or for data aggregation. Non-blocking loads using asynchronous data requests drastically improve the perceived page load time and provide a nonblocking loading of the page.
UI Tier	The inbuilt caching of web servers should be leveraged for better performance and scalability.
Business Tier	Designing logical layers on the same physical tier to reduce the number of physical nodes while also increasing load sharing and failover capabilities.
Business Tier	Leverage design that uses alternative systems when it detects a spike in traffic or increase in user load for an existing system.
Business Tier	Stateless transactions and requests makes the application more scalable. Designing applications using stateless session beans improve the scalability.
Business Tier	Business logic needs to be loosely coupled with the web tier, hence deploying the application components on the separate node is easier. SOA provides scalability at the integration layer by loose coupling.
Database Tier	While retrieving data from systems, such as a database or web services, it is recommended to batch requests to reduce server round-trips. Most of the database APIs and object-relational mapping (ORM) frameworks provide batching functionality.
Database Tier	Use connection pooling for database and resource pooling improves the scalability of applications. Resource pooling, such as database connection pools, is for maintaining multiple logical connections over fewer physical connections and then reusing the connections, bringing in more scalable efficiency.
Database Tier	Partition data across multiple database servers to improve scalability and allow flexible location for data sets.

STATIC ANALYSIS – CODE ANALYSIS

Static code analyser's assesses quality, in terms, degree of compliance with the coding practices of software engineering that promote security, extensibility, reliability, and maintainability. Static analysers find weaknesses in the program code that might lead to vulnerabilities. Static code analysers analyse the source code for specific defects as well as for compliance with various coding standards and coding guidelines. The tool identifies security vulnerabilities and hotspots during development and catch these critical issues. Fixing these flaws during implementation phase can reduce the number of builds necessary to produce an optimum and secured product and educate the development teams about coding practices and guidelines. Static code analysers review the source code to detect common bad practices, catch bugs, and make sure the development adheres to standards and guidelines. Most static code analysis tools define a series of rulesets (100+ rules) that identify different categories of issues in the code, for example: programming errors, coding standards, violations, and security vulnerabilities.

The challenges of modern software systems converge ultimately to their architecture. As systems become more complex and huger, their architectures assume ever greater importance in managing their growing coherence, reliability, and integrity. When architectural integrity is compromised, the probability for a serious operational bottleneck increases dramatically. Interactions among layers and subsystems will become increasingly more complex to articulate. Software Composition Analysers look inside to identify architecture quality issues. The analyser's read, analyse and semantically understand all major kinds of source code across all layers of an application (GUI, logic and data). By analysing all tiers of complex software, critical application health metrics like robustness, maintainability, transferability, flexibility, performance, or security can be measured and compliance to the best prac¬tices can be assessed. The analyser's look at the application from a static viewpoint but are able to simulate how the application will run, connecting all pieces of the puzzle, looking across different languages and databases. Hence, analysers are able to perform analysis of the entire application or system as to its health.

DYNAMIC ANALYSIS – RUN-TIME ANALYSIS

Dynamic Analysis are a set of processes and tools to ensure that application remains highly available and responds to user requests within an acceptable time limit. Monitoring tools help to achieve these goals by monitoring metrics such as response time, memory, network bandwidth, IOPS, and CPU time. The next generation tools that are based on cutting edge technologies like machine learning and AI provide

ways to diagnose, triage, and resolve issues and bottlenecks in applications and infrastructure. The aspects that are critical interms of application monitoring or APM are monitoring metrics, tracing and logging.

- Application Monitoring: In terms of monitoring, the metric is a quantified measure that can be leveraged to understand the status of a service, process or an application. Metrics are often compared to a defined baseline to analyse the application or process's status and its overall heath.
- Transaction Tracing: A trace aka transaction tracing is used to understand the complete journey of an end-user request as it travels through all components, services, processes, and different layers of a business application.
- Application Logging: Log files are automatically created by an application and they provide information about end-user behaviour and events that took place in the application life cycle from start-up till the shutdown.

Application monitoring provides detailed visibility into the performance, availability, response times, and user experience of application and its underlying infrastructure. The application monitoring helps not only to monitor but rapidly triage, diagnose, and resolve issues leveraging cutting edge tools and technologies. Application monitoring tools collect, stores, and analyse the necessary data and metadata for troubleshooting, optimizing performance, root cause analysis, and final resolution. They typically rely on different types of instrumentation and profiling processes to provide real-time insights into the application health and its status. When performance exceeds automatically defined thresholds, application teams are notified and then can drill down contextually to trace transaction and performance issues across the distributed infrastructure for triage and resolution.

Few most critical application monitoring metrics include:

- Performance Monitoring: Measures the average response time for end-user interactions to check if application performance is affecting the speed.
- CPU usage: Monitor CPU usage, disk read/write speeds, IOPS, and memory to see if application performance is impacting these key parameters.
- Application availability and uptime: Measures whether the application is online and available and in good health to end-users. This is also leveraged to determine compliance with an organization's SLA.
- Request Rates: Measures the amount of traffic received by the application to identify any significant increase, decreases or coinciding users.
- Error rates: Observes how the application degrades or fails at the software level.
- Discovery: Counts and monitors the servers, applications, services, and processes that are running at any one time.

Organizations create rules, so the monitoring tool alert them when a problem arises or when an application's performance metrics dips in a specific area. They can also prioritize applications based on business criticality. In virtualized deployments, APM tools can help monitor application servers to ensure that they comply with an SLA. Cloud introduces a host of additional dependencies on application performance. There is cloud application performance monitoring, which focuses on tracking the performance of applications based on private or hybrid cloud deployment models.

Key Characteristics of the Modern Monitoring Systems:

- Anomaly Detection. Anomaly detection capabilities in monitoring tools can automatically alert users when metrics deviate from the thresholds, assess their impact, all without human intervention.
- Correlations. Correlation engines go a level deeper and compare all parameters that contribute to metric outcomes. They analyse and measure subtle changes in one or more business metrics.
- Root Cause Analysis. Root cause analysis engines go even further and suggest possible causes of a deviation from the normal benchmark of a business metric or a group of metrics. These engines articulate the cause from historical correlations, or they provide IT-users with tools to assist them localize a cause by comparing multiple correlations.
- Automation: After detecting an anomaly, the product will determine its root cause, suggest a

remediation, and predict the future events. They may even suggest ways to optimize the business process to eliminate future incidents.

TOOLS & METRICS

One must recognize which profiling & optimization tools might help in analyzing the root cause of performance issues in the application. One will need both profiling, optimization and monitoring tools to identify and triage the performance hotspots. Every tool and profiler have its own features and advantages. Few of these tools have characteristics that allow you to get continuous feedback on the performance of code and SQL. These tools provide impressive features that help visualize the entire application and its components, interfaces, database into a dashboard like views that help you deep-dive in and triaging the issues. The following tables provides the different tools can leverage for this purpose.

Tooling Options	Description	Findings & KPIs
As-Is State Analysis		
Static Analysis (Code)		
SONARQube, CAST Highlights, Fortify, Veracode, Checkmarx	Static code analysis is the best way to ensure code quality as it checks the code thoroughly line by line and gives developers the opportunity to correct flaws before the software goes live. This flags bugs, code smells, security hotspots, and vulnerabilities. Though which, we can also trace and tackle performance issues.	Bugs, coding standards, Security hotspots and Vulnerabilities
MySQL Query Analyzer,MySQL Query Explorer, Workbench	This tool will provide insights into optimum queries, errors in SQL, severity of the errors, inefficiencies etc., through which recommendations can be provided.	Optimum queries, errors in SQL, severity of the errors, inefficiencies
Dynamic Analysis (Run-Time)		
KCashGrind, QCashGrind, CallGrind, Xdebug,	This is a profiling and instrumentation tool and will provide detail interms of call graphs and tree maps, visualization of the APIs/Calls	Call graphs and Tree Maps Visualization
App Dynamics, Dynatrace, New Relic, CA APM	Monitor software services and applications in real time — collect detailed performance information on response time for incoming requests, database queries, calls to caches, external HTTP requests, and more.	Transaction Tracing, MySQL Metrices, Query Throughput, Query Execution Performance, Connections, Success/Errors, Buffer Pool usage
MySQL Tuner, Releem	This is a tool to review MySQL installation and make adjustments to increase performance and stability.	Performance and Stability

Table: Tooling - Static & Dynamic Analysis

This is highly advisable that every application one builds should have the right set of performance testing and profiling tools to ensure that the application runs smoothly without a glitch. There are a variety of tools available that can monitor and profile your application's performance as listed in the above table.

RELATED WORK

[1] & [2] This paper describes PASA, a method for performance assessment of software architectures. PASA uses the principles and techniques of software performance engineering (SPE) to determine whether an architecture is capable of supporting its performance objectives

[3], The goal of this paper is to classify the performance prediction and measurement approaches for component-based software systems proposed during the last ten years.

[4], This paper proposes a new approach to improve the performance and scalability of HPC applications on Amazon's HPC Cloud.

[5], In this paper, we build performance models for applications in virtualized environments. We identify a key set of virtualization architecture independent parameters that influence application performance for a diverse and representative set of applications.

[6], This paper presented the business monitoring and its trends. These systems detect anomalies, identify correlations, and display potential root causes.

CONCLUSION AND FURTHER WORK

The architecture of a software system is the primary factor in determining whether or not a system will

meet its performance and other quality goals. Architecture assessment is a vital step in the creation of new systems and the evaluation of the viability of legacy systems for controlling the performance and quality of systems.

Software performance is not achieved in isolation. Performance objectives must be balanced with other software quality concerns including reliability, availability, extendibility. Sometimes, these objectives conflict when architectural features have opposing effects on different quality attributes. For example, redundancy may increase availability but negatively impact performance. Identifying the areas of the architecture where conflicts occur and quantifying their effects makes it possible to find a workable compromise. As with performance objectives, to evaluate the effect of architectural decisions on qualities such as modifiability or reliability, it is important that the requirements for these attributes be stated precisely.

This paper presented an approach for performance analysis and assessment of software systems. It described the methods we use in a variety of application domains including web-based systems, financial applications, and real-time systems. It described the key pillars of the approach which consists of As Is Analysis, Static Analysis and Dynamic Analysis. As next steps, research should be conducted on the systems that automate things that humans can't do and augment what humans can with real-time recommendations and a good degree of automation.

REFERENCES

[1] PASASM: A Method for the Performance Assessment of Software Architectures Lloyd G. Williams, Connie U. Smith

[2] PASASM: An Architectural Approach to Fixing Software Performance Problems Lloyd G. Williams, Connie U. Smith

[3] Performance Evaluation of Component-based Software Systems: A Survey Heiko Koziolek

[4] Improving HPC Application Performance in Public Cloud Rashid Hassani, Md Aiatullah, Peter Luksch

[5] Application Performance Modeling in a Virtualized Environment Sajib Kundu, Raju Rangaswami, Kaushik Dutta, Ming Zhao

[6] Business Monitoring Systems Using Machine Learning to Analyze Business Metrics Wayne W. Eckerson

[7] [7] The Architecture Tradeoff Analysis Method Rick Kazman, Mark Klein, Mario Barbacci,Tom Longstaff, Howard Lipson, Jeromy Carriere

AUTHORS BIOGRAPHY

Sameer S. Paradkar is an enterprise architect with more than 20 years of extensive experience which spans System Integration, Product Development, and advisory Organizations. Sameer works as an SME on architecture modernization and transformation initiatives. He has worked on multiple digital transformations, engagements and large complex deals in North America, Europe, Middle East, and ANZ regions that presented a phased roadmap to the transformation maximizing business value while minimizing costs and risks. Sameer is certified and competent in different methodologies and frameworks including: TOGAF, NGOSS (e-TOM & SID), ITIL, COBIT, Agile, Scrum, DevOps, Scaled Agile Framework – SAFe and Business Capability Modeling. Sameer is part of the Architecture Group in AtoS. Prior to AtoS, he has worked in top tier SI and consulting organizations.