Jindo: Smart Microservice Monitoring and Development Tool

Mehmet Göktürk
Computer Science Dept
GTU
Gebze Turkey

Imran Kazdal
R&D Center
Borsa Istanbul A.Ş.
Istanbul Turkey

Ahmet Faruk Biskinler
R&D Center
Borsa Istanbul A.Ş.
Istanbul Turkey

ABSTRACT
Recent developments and programming trends made microservice architecture quite a popular approach for enterprise information systems. Classical monolithic mega-applications are slowly being replaced with microservice based container clusters. Advantages such as scalability, maintainability an suitability for continuous development and harmony with agile software teams make them a favorable choice. Yet there are some disadvantages regarding their operation, management and development. Especially in large institutions where large number of microservices are developed and put into service, monitoring them in runtime becomes burden as well as keeping them in harmony with each other. Developers may face serious difficulties sometime after an enterprise microservice transformation.

Lack of adequate monitoring, difficulties in understanding the underlying program control logic can cause serious problems and disruptions as well as unacceptable performance. Furthermore, microservice development is not enterprise wide controlled process yet. In this work, an integrated enterprise scale microservice monitoring and production system has been introduced. Smart features relying on machine learning techniques are used to monitor performance of microservices predictively on a heterogenous enterprise scale environment. Moreover, through a development control and template code generation feature, microservices that are being developed within the institution are put into tighter control. The system named as Jindo, included additional features related to security and maintenance as well. The results obtained suggest thatsystem managers and developers were affected very positively and enterprise application performance can be enhanced through Jindo system.

General Terms
Software creation and management, Software post-development issues, System administration

Keywords
Microservice, service monitoring, model driven development, ide, software development

1. INTRODUCTION
Microservice-based enterprise software development and operation recently become a preferred choice due to widely recognized advantages. Since development, commissioning, operation, debugging and alsoscalability features of monolithic large structures are relatively poor, recent literature and experts suggest a shift towards microservice based architectures. Initially, service oriented architectures (SOA) were seen as primary solutions for interoperability and scaling in enterprise solutions. Service oriented architectures opened the roadwork for further smaller services where development, operation and maintenance can be divided and assigned onto smaller structures both as teams and as hardware[1].

Popularity of scalable virtualization container architectures made smaller services be more viable solutions for enterprise software systems. The term microservice architecture is then used to indicate small scale, usually single or simple tasks that are associated with primary functions of monolithic applications. It is seen that popularity of microservices in heavily used enterprise software systems is significant yet is brings serious new challenges to development[2][3].

Wide acceptance and positive perception of microservice architectures among developer community also leads to several misconceptions as well. They are usually seen as lifeguards where hard to maintain projects are already causing terrible times with developers begging for complete transformation[4]. Such cravings for microservice transformations by developers mostly ignore structural changes that are necessary to implement in application design, development, operation and maintenance. Traditional complexity in monolithic application is then moved onto network layer and interworkings of the microservices by microservice transformation, rendering traditionally easy to track things very difficult such as performance tuning, security, parameter passing, database design, code design and so on[5][6][7].

Yet, fast moving enterprise software development bandwagon, with many teams working together, tight deadlines, serving thousands of customers need a viable change in development perspective and currently microservice transformation is one of the solutions as taken as future path to go[8].

Moreover, rapid change in enterprise needs, lack of qualified software developers, methodologies that tend to eliminate software developer personnel errors, DEVOPS systems put additional pressure on large scale organizational software development[10][11].

In this study, an integrated smart system is developed in order to address the needs of large enterprise with high number of microservices (over 100) working in a heterogenous environment. Such organizations require continuous development of new microservices, continue migration of old monolithic architectures, moving to container based architectures and require careful planning and monitoring of all the systems in an efficient and secure manner. The system developed is named as Jindo featuring consolidated management, monitoring and development framework of enterprise wide microservice solutions. It is also empowered with smart predictive performance monitor enabling to take

measures such as scaling up and restarting in advance before service interruptions occur.

The rest of the paper is organized as follows. First the issues in Microservice transformation is mentioned. Then implemented Jindo architecture is presented. Main features or implemented solution are discussed. Predictive algorithms are introduced. Performance of predictive monitoring and management features are presented. A conclusion and discussion are added to the end.

2. MICROSERVICE TRANSFORMATION

Transformation from procedural programming languages to object oriented languages posed significant challenges at the time of transitional years. Many developers were hesitant since control structures were less visible to them once objects were responsible from the execution of traditional program logic[12].

A similar transformation is now ongoing with the microservices. Many crucial parts of the program structures that were classically wired into object methods and object behaviors are now separated into different microservices. Although it may seem quite reasonable and fruitful at the beginning, such transformations bring several problems associated with them. First, application logic is pushed towards network layer where relations between object classes are somewhat mapped to relations between different microservices, with quite a lot of exceptions. Therefore entire application needs to be reviewed and restructured, considering shift to network communication from object communications where simple pointers and stack would make lightning speed transfer of data between objects[13][14]. With microservice architectures, “processing” and “transfer” costs become main competing parameters while dividing application into microservices in addition to separation of concerns principle.

3. JINDO ARCHITECTURE

Jindo Architecture combines fundamental needs of enterprise information systems in microservice development and deployment under one roof. Main categories of functions of Jindo are given as follows:

- Microservice Management and Monitoring
- Automated Template Based Code Generation
- Security and Authorization
- Logging and Predictive Analytics

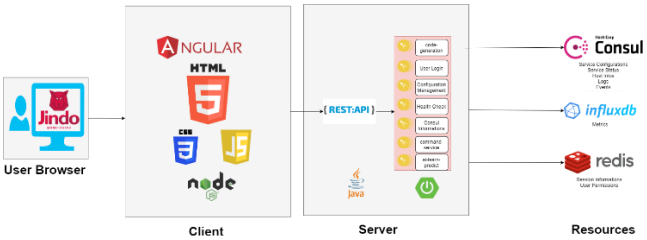

Figure 1: Jindo Microservice Monitoring System Structure

4. SMART MONITORING METHOD

Monitoring task is one of the fundamental tasks in enterprise software systems operations. Classical server monitoring tools have been widely used for a quite amount of time. Moreover, runtime technologies and virtualization paradigms lead to further enhancements in monitoring systems. With the introduction of Service Oriented Architectures, monitoring “services” rather than “servers” became the primary focus[15].

One aspect of microservice migration is providing facilities for efficient monitoring. A "Service Record Database" is usually created for the central management of microservices. In this database, service names, servers that services are running on, addresses and port of the server they are running on are stored. The services are then extracted from this database and shown on a visual topology map or in the form of a display list.

Various visualization techniques can be utilized for systems management and developer staff in order to enable better understanding both active microservices and those under development in an institution[16].

Through this strategy, as studied in previous research, "Visual Topology Map & Chart Module" has been created within the scope of this work. Thus, it became possible to interpret relationships between microservices, comprehend and improve system and program architecture, avoiding possible errors and bottlenecks.

Microservices are individually run yet dependent program units. Conveying dependency information to user has been indicated as crucial in monitoring microservices. In Jindo, dependency topology screen between services can be displayed. For this purpose, NodeJS, Angular and vis.js libraries were used to display relevant dependency information as a graph with nodes as microservices. In addition to dependencies between services, dependencies of the file system and the database have been shown, including additional data which developers may need in an assimilable structure.

A service management module was developed with a structure that can automatically register and provide management functions for the server on which the services are running. Throughan automatic service discovery feature, each newly added microservice to the microservice record pool automatically updated in the visual topology map using the open source “Consul” infrastructure.

Figure 2: Jindo Microservice Monitoring Interface

Primary functions that access to up-to-date information and actions to be taken to all services that are automatically registered in the details section through the interface, are given below:

- Listing microservices
- Access to service detail information
- Filtering operations
- Identifying inter-service dependencies and inferring impact analysis
- Display and save visual topology map of the services

- Follow-up of service health status changes
- Configuration management
- Alarm monitoring and management
- Log monitoring and management
- Action management (Start, Stop, Kill etc.)

4.1 Service Log Module

Post event and predictive analysis of microservices require logging performance parameters accordingly. With the log module developed in Jindo, fundamental log operations can be performed in accordance with "Kibana" system. Integration with Kibana is provided by clicking on a log message on the screen to obtain further detailed information including events that happen before and after.

The logs of the microservices are obtained by reading the data from the log files with the "Logstash" application installed on the relevant servers, processing efficiently with the Elasticsearch application and displaying on "Kibana" system. While there were microservice application logs, microservice request logs on Kibana, they can be processed in any log analyzer as any file kept on the server.

Furthermore, the logs shown on Jindo application log screen are also integrated with Consul and are able to receive Consul events belonging to the microservice of interest.

If the operator wants to examine more detailed information about an error log, he can see log of the microservice in the relevant time interval in detail by clicking on the log message in the interface. At this integration point, the Jindo application directs the Kibana address with an address that has been processed accordingly.

Further achievements on top of log data can be obtained through implementation of predictive log analytics. Jindo, for this purpose has been equipped with machine learning tools that can estimate oncoming problems before they happen. A specially developed microservice proactive anomaly detection/ action module of Jindo provide anticipation of microservice metric data with intelligent learning algorithms. It can be used in detecting errors and / or alarm situations before they occur and take relevant actions in advance. Estimation is performed using collected metric data via clustering and regression models developed using past data.

Service metrics are recorded in a time series database by means of java agents that are defined in JVM parameter sectionof microservices that are of interest. Situational estimations of metric data are performed with Jindo AI module. The estimates are then transferred to the graphics user interface and action engine via metric-entity-service.

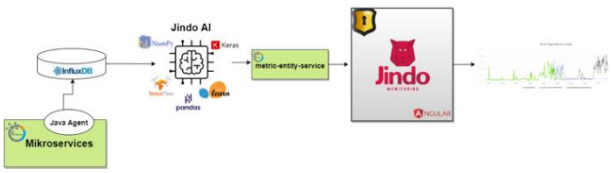

Figure 3:Proactive Monitoring Engine

For example, the JVM parameters of file-download-capability-service were arranged and metric data were saved to InfluxDB as follows:

> "-javaagent:'/opt/agent/java-influxdb-metrics-agent-0.0.6-SNAPSHOT.jar=servers=10.57.2.16:8086,database=int2,interval=1,tags.host=yconnd01,tags.ip=10.57.2.139,tags.microservice.name=file-download-capability-service'"

Using Java agent method,various metrics such as "metrics", "lang", "nio" etc. that are already included as java library metrics can be obtained easily. For example, HTTP request metrics on the service, "SystemCpuLoad", "HeapMemoryUsage.used", "NonHeapMemoryUsage.used", "MemoryUsed" etc. are some of the metrics that wereobtained and utilized by Jindo.

In order to construct prediction models, "stress load" data was first created by performing load tests on the services in test environments and the predicted estimations were compared against these. These are discussed in results section of this work. Since human system operators and managers are highly interested in current condition and estimated conditions of these metrics, a metrics monitoring user interface has been created. A metrics monitoring screen has been designed to give system monitoring person a deeper understanding of the enterprise information systems under operation. This user interface is based on interface in Figure 4.

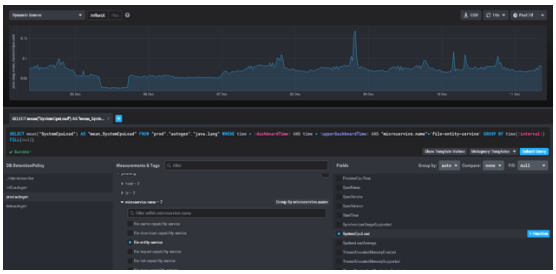

Figure 4:Jindo Microservice Metrics

Based on the results returned from the Jindo AI module, proactive actionsare then taken according to the threshold values entered in the Alarm Handle Service. A continuous round robin running tracking engine tracks all the parameters that are set in tracking mode by comparing them first with preset thresholds. When found appropriate, scaling operations are then performed on the container by calling "Alarm Handle Service" and"Container Manager Service" respectively.

Proactive monitoring function is enhanced by adding machine learning properties to the module, in addition to the simple threshold detection. Various machine learning methods are performed and compared in order to label outlier performance metrics. They are validated against labeled data that are collected during stress testing of enterprise services in test environment. Results of proactive algorithms are discussed in detail in section 7 "performance results".

4.2 HealthCheck Module

As dealing with microservices, tight coupling of various elements in program structure yet being run as separate services makes root cause analysis difficult. This is sometimes due to incorrect design and transition to microservice architecture from a monolithic program[17].

In order to find the root cause of problem and make impact assessment in case of presence of any error in an enterprise system, Jindo displays health status information of the services in a visually understandable format. With the data obtained from Consul helper application, the health status, state changes, relations with each other and the number of requests they made to each other per second of all microservices being used and newly developed are shown on the visual topology map that is drawn by Jindo user interface.For example, a smoothly working microservice and the server (s) that it runs on are displayed in green on the

topology map. If at least one of the servers where the service is running has an error, it will appear in yellow and a damaged microservice(s) are displayed in red. This allows operator to identify relations that are associated with error and find and fix the problems as soon as possible and take necessary actions. Color coding scheme of health check user interface is given in Figure 5.

Figure 5:Health Check Color Coding

5. JINDO DEVELOPER TOOLS

Aside from monitoring and management functions, enterprise organizations are in need of tools that guides and keeps production of microservices under control due to several reasons. Jindo developer tool module aims to help microservice development in organization with the knowledge and expertise that are gained from monitoring and management data as well as current state of the art.

In the module developed, it is aimed to keep the microservices creation and management behavior of the developer staff under control as well as automatically perform many tasks that were considered as repetitive, routine and boring for developers. These repetitive “bureaucratic tasks” in fact cause significant portion of common errors and disintegration of programming discipline.Similar to the traditional monolithic software development languageswhere similar types of template-based graphics libraries and add-ons supporting framework components are integrated into development environments Jindo tries to integrate microservice creation and templates into development environment so that routine parts of development process can be regulated and streamlined. With a similar approach, it is aimed to obtain significant advantages by applying to the process of in-house development of microservices throughout enterprise.

5.1 Automatic Code Generator Module

Today’s most new generation developers do not want to write code that is routine and repetitive, without creative features, less interesting and challenging tasks can be automated. In order to alleviate burden of next generation developers, automatic code generator module provides opportunity to the software developer to model the current microservice project through help of auxiliary functions and interface to the finest detail. It then allows developer to save it and manage it. A template support tool allows developer to generate desired microservice code by selecting from existing template set.

The templates provide support throughseveral technologies incorporated into Jindo module and expect developer to answer several interactive questions. (MongoDB, Oracle, Frontend, Backend, etc.). Developer is then asked to select project types and fill other constraints and parameters.

After completion of initialization questions, the microservice developmentproject is automatically derived and presented to the software developer for detailed development work to continue.With the help of Jindo production tool developed in the study, it is ensured that the software developer downloads the derived code to its own development environment and work on it there and continue to work by customizing the derived code and submit through enterprise software development environment. In addition to the reduced burden of developers, improved software reuse is one of the significant advantages. This is accomplished in accordance with catalog management function which is discussed in next section.

The module was initially planned to accomplish automatic code generation approach to transform around 300 microservices into a new structure in the enterprise. It is expected to be adopted by sister institutions and then planned to be launched as a commercial software development tool. With the help of module, analysis, development and API documentation will be standardized and automated within the enterprise. Since transformations of existing microservice software base is usually required, the module supports transformation by providing a unified generation interface and codebase.

5.2 Catalog Management

Cataloging and classification of services is a feature that developers want most of the time in enterprise software development due to large basis of existing software. Through cataloging and classification, software reusability can be increased and development processes can be improved. Althoughall microservices can be developed and managed relatively manually, it may take a significant time for developers to achieve an adequate level of developer situational awarenessas the number of microservices increases. The advantages of Jindo providing catalog management functions are reported by developers within the enterprise and it is considered as one of the strengths of the developed system.

In addition to basic management functions, scope definitions are also used to narrow down developer context into the target microservice. With the module, it is ensured that only microservices in a certain scope suitable for the developer’s context are highlighted without a large list of all microservices.

The management, tracking and reporting of the projects through cataloging include, infrastructure components, technical-functional details (group / person information that developed the project, application IP, port information, database, URL, user name, password information, etc.) are recorded in catalog data.

5.3 TeamCity Integration

TeamCity is a build management and continuous integration and continuous delivery (CI/CD) server from JetBrains company. Jindo allow integration of microservice development and deployment with TeamCity software. Through this integration, service related plans can be created on TeamCity. In this way, service can be deployed to the desired environments automatically. Deployment package release can be released to the relevant environments.

First of all a service specific deployment plan can be created. This plan can be accessed and edited at any time. In this way, the deployment plan of all services is kept in a common, easily accessible and manageable place.

Secondly, services can be easily deployed to the desired server. In case of any error during deployment, TeamCity will give a warning and indicate that the deployment has failed. Since the outcome is automatically visible from Jindo environment, proper use of TeamCity platform is also provided.Lastly, version control and related information regarding versioning of the service that is deployed can be obtained through Jindo-TeamCity integration.

5.4 GIT Integration

GIT is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. It's a widely popular tool among developers and software developing institutions. It can be locally deployed or public GIT server can be used wo empower collaborations. Primary goals of GIT system include speed, data integrity, and support for distributed, non-linear workflows. Using GIT integration feature of Jindo,a project can be transferred to GIT system automatically. In this way, the integrity and version of a project can be tracked through GIT.Some of therelevant advantages of GIT are therefore transferred directly to Jindo system as follows:

Providing the opportunity to manage and organize all services in a common area.
Making observation of the codes of the service possible.
Making collaboration with external actors possible through GIT.

6. SECURITY FEATURES

Microservice transformation in enterprise environments may create new security issues which are not previously common. In a traditional monolithic application, objects and methods and other relevant functions share the same memory space with the same security privileges under an operating system security scheme. However, when these programs are transformed into separate microservices, functions, threads or child processes that were in the same program memory become "foreign" to each other[18][19]. This immediately requires attention on security and integrity mechanisms that allow cooperation between microservices which once were methods/functions of the same monolithic application [20][21][22].

Jindo system defines special approach to security problems of microservices through several authentication features.

6.1 User Authorization

When a user log into the application incorporating microservice, authentication of the user is performed with security and authorization module. The specified verification mechanism has been implemented using "Spring Security" (JWT) technology and LDAP integration in Jindo.

Spring Security ensures that the application developed becomes more secure by providing its own authentication and authorization models. In addition to security, it is useful to developers since it can be easily integrated into any web application and supports multiple types of authentication methods. Among these, In-memory, DAO, JDBC, LDAP etc. can be given as primary examples. Spring Security has an infrastructure that can be integrated separately.

Jindo can allow developers to utilize LDAP authentication methods. LDAP (Lightweight Directory Access Protocol) is an open source directory service protocol with a highly flexible architecture that is generally used for authentication. It is the communication language used by applications to communicate with other directory service servers. Directory Services stores users, passwords, computer accounts and shares this information with other entities on the network.

Due to its advantages, Spring Security technology was used in Jindo Security and Authorization Module and authorization mechanism was established by integrating with LDAP among the supported authentication types. In this way, users can log in to the application by authenticating with their LDAP username and password. The user authentication to login to application is given in Figure 6 and explained in step by step below.

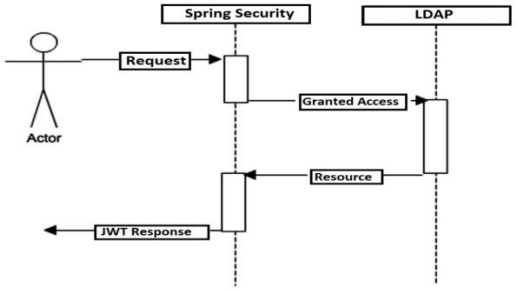

Figure 6: User Authentication

1. A login request is sent to the microservice side, which is integrated with LDAP, with the user name (or mail) and password on the client login page.
2. With the information sent to the microservice side, the user name and password are verified over LDAP.
3. If the username and password are valid, the token is created with a pre-determined secret key and the token turns to the client side.
4. After logging in, all other requests of the user in the application are verified over the token.

6.2 Authorization BetweenMicroservices

Transformation of monolithic applications to microservices result in microservices calling each other frequently. As described previously, it becomes necessary to introduce a security mechanism. In Jindo, as microservices call each other, OAUTH2, a standardized protocol used for authorization between services is utilized. In case a service calls another service, the following checks are carried out step by step:

1. The microservice to be called receives tokens from the OAUTH2 server with its authorized user.
2. The service to be called makes a request to the service with the token received.
3. The called service checks the authorization of the received token through the OAUTH2 server.
4. If the requested token is authorized, a response is returned to the API.

Between microservice authorization model used in Jindo security approach is shown in Figure 7.

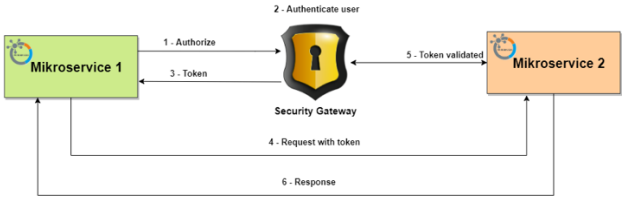

Figure 7:Microservice Authorization Model

Furthermore, authorization levels of microservice users can be defined by authorized users through the Jindo application.

With these separate authorizations, each API of each service can be accessed based on authorization.

7. PERFORMANCE RESULTS

It has been reported by over dozen of microservice developers that Jindo enterprise system helped them in development of new microservices or making new refactored versions of existing ones. Operational staff also provided very positive feedback about the user interface and reported benefits after using initial release versions of the monitoring user interface.On the other hand, the most interesting part of the Jindo system to be evaluated is the power of predictive monitoring module. By completion of Jindo microservice monitoring and management tool, it was necessary to evaluate methods used.Through the metrics collection module data has been collected from over hundred microservices. As a result of the load tests, test data sets were obtained. The data are kept in InfluxDB storage database. Machine learning algorithms, clustering and regression models are run on these metrics that are recorded on the order of seconds. Following sections briefly describes the results that are obtained using each method.

7.1 Clustering Models

In order to be able to determine an outlier, it is hypothesized that clustering algorithms on time series data can be utilized as a simple approach. K-means model, one of the unsupervised learning models, was first chosen as the clustering algorithm. K-means model was run on CPU, Memory, and request metrics.

The model is regularly rebuilt in specified periods.K-means model can mark which dataset the new point belongs to in a very short time as response time. Elbow method was used to determine number of clusters. The results of Elbow method yielded 6 clusters (WCSS 1.3, 6). A real time visualization method is developed for metrics that come from the metrics collection module. The outcomes were visually successful but further models were suggested due to classification performance.

7.2 Regression Models

In regression model approach, models were studied on collected data using LSTM, XGBOOST, Random Forest, AR, ARIMA and HWES models. Due to the correct estimation rates and speed, workwas continued with AR, ARIMA and HWES models after initial trials.With XGBOOST, the mean square error RMSE value obtained was 0.030330. Generally, the estimates did not come close to the real values. Random Forest was used and Mean square error RMSE value was calculated as 0.026360.An alternative Holt Winters model has been tested by changing the seasonal periods parameter. Examples from the work done were visualized. Mean square error is 0.0004 and 0.0001, respectively. The estimations made were close to the real values as seen in Figure 8.

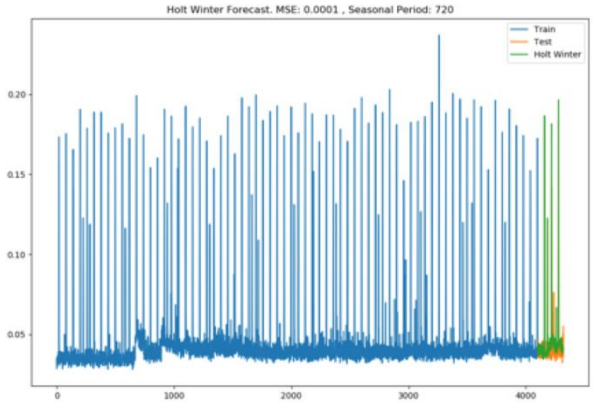

Figure 8:Holt Winters Model Results

Then ARIMA model estimation parameters were changed to observe the mean square error values.The average of the error decisions was calculated as 0.0009 in ARIMA. Accordingly, p, q, d values were chosen as 10.0.0.

The graph in Figure 9 has been obtained by giving ARIMA model q, p, d parameters respectively 10,0,0. Blue colored data represent test data, and orange colored data represent data estimated by Arima model. Each step in the graph has been tried to be estimated in seconds. The mean square error (MSE) was calculated as 0.00041.

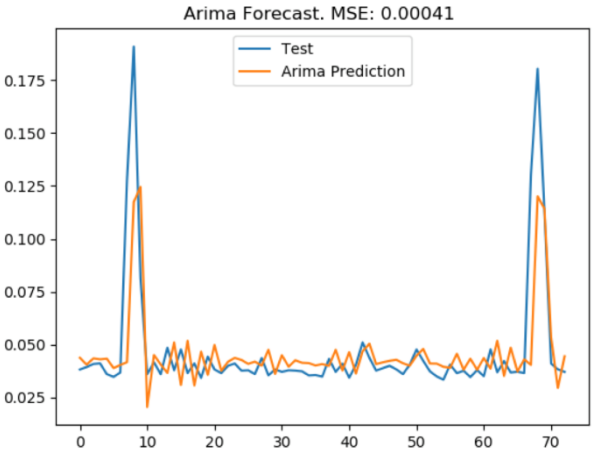

Figure 9:ARIMA Model Results

ARIMA model results were then transferred onto monitoring and smart prediction interface that is shown in Figure 9. For comparison AR (Autoregression) model was also tested. Autoregression Model parameters were automatically selected by the model and estimations were made.

In Figure 10, blue colored values represent test data and red colored values represent estimates on test data.In the study, the mean square error value (MSE) value was calculated as 0.0009.The real-time predicted data in the Jindo application with the AR model are shown in Figure 11.

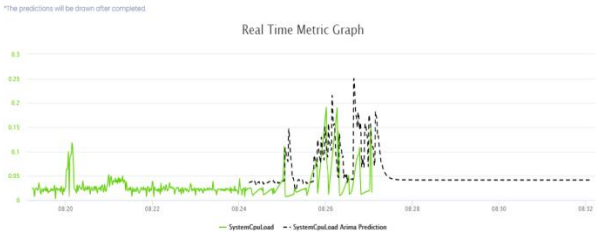

Figure 10:ARIMA Predictor on Monitoring Interface

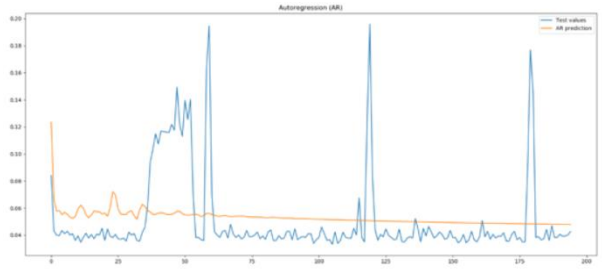

Figure 11:AR Model results

After testing with above models AR, ARIMA and HWES models were found to provide satisfactory performance. The precision and accuracy rates in the second-order estimation results were found to be high in the ARIMA and HWESmethods. Yet, AR model, was chosen for its speed and accuracy. In figure 12, prediction interface is shown with green lines representing actual data and dashed lines representing estimated values. Therefore AR Predictor on Monitoring Interface was selected for initial operations.

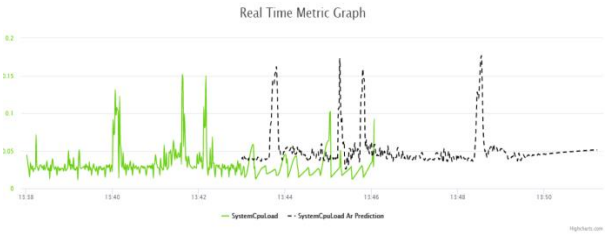

Figure 12: AR Predictor on Monitoring Interface

8. CONCLUSION

Given the requirements of high availability and low latency, microservice transformation has been in place for quite some time. However, large number of heterogenous microservices made off-the-shelf tools unusable. If, a software is going to be developed from scratch, then it would be much easier to work with available development platforms or develop a simpler solution. However, with an enterprise of this size, large number of microservices are already in service, each of them belonging to different generation of paradigms and rules.

Development of Jindo allowed enterprise system monitoring engineers and developers to get the big picture of what is running for the whole operation and empowered them for writing better microservice code.

Since they were interrelated, not only for monitoring, but also development process was considered in the work. Effortswere made to integrate microservice production tool with commonly used operating environment and developer tools: Docker, Kubernetes, Consul, Kibana, TeamCity, SVN, GITwereamong the leading ones.

Finally, a design was made to expand the microservice standard API specification to support the API / documentation of the services, and development and sample integration libraries were provided based on this new specification. In addition to the dependencies between the services, it is aimed to indicatethe file system and database dependencies. For this purpose, a visual module was created and the relationship between services was shown in the user perspective. This way, the assimilation of the system architecture by future software development staffwasimproved. Initial reports were significantly positive from within the enterprise developer community.

The use of Jindo can improve assimilation of the system architecture by future software developer staff and provide situational awareness to the developers.

We believe that Jindo or similar tool will become essential for large scale enterprises where microservice transformation and development continues. We hope to create a commercial version in coming years for others to use.

9. DISCUSSION AND FUTURE WORK

During the work performed in the enterprise, we realized that majority of problems are caused by poor understanding of underlying structure of problems and poor understanding of how microservices work by developers and system managers. Microservices do not simply work like servers. They are interconnected, functions are separated with different database scopes etc. Therefore, system monitoring paradigm does not simply alleviate performance problems of microservices. There is a network layer, different performance metrics and different latency issues. These have to be addressed very well by developers and also have to be enforced by system managers.

One of the future works that are required after this study is to make developer user studies in order to understand how such a tool can improve programming quality and programming performance of developers and how they are related to microservice runtime performance and probability of bugs being present. We are planning to make a companywide formal questionnaire study to obtain formal initial results to assess positive outcomes. .

10. ACKNOWLEDGMENTS

This study has been prepared in line with the studies being carried out in a part of TEYDEB-1501 Project named “Jindo-New Generation Proactive Microservices Management and Development System” numbered TÜBİTAK-3190614.

11. REFERENCES

[1] Dragoni, Nicola, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. "Microservices: yesterday, today, and tomorrow." In Present and ulterior software engineering, pp. 195-216. Springer, Cham, 2017.

[2] Hassan, Sara, Nour Ali, and Rami Bahsoon. "Microservice ambients: An architectural meta-modelling approach for microservice granularity." In 2017 IEEE International Conference on Software Architecture (ICSA), pp. 1-10. IEEE, 2017.

[3] Rademacher, Florian, Jonas Sorgalla, and Sabine Sachweh. "Challenges of domain-driven microservice design: a model-driven perspective." IEEE Software 35, no. 3 (2018): 36-43.

[4] Pahl, Claus, and Pooyan Jamshidi. "Microservices: A Systematic Mapping Study." In CLOSER (1), pp. 137-146. 2016.

[5] Asik, Tugrul, and Yunus Emre Selcuk. "Policy enforcement upon software based on microservice architecture." In 2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 283-287. IEEE, 2017.

[6] Haselböck, Stefan, Rainer Weinreich, and Georg Buchgeher. "An Expert Interview Study on Areas of Microservice Design." In 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), pp. 137-144. IEEE, 2018.

[7] Naily, Moh Afifun, Maya Retno Ayu Setyautami, Radu Muschevici, and Ade Azurat. "A framework for modelling variable microservices as software product lines." In International Conference on Software Engineering and Formal Methods, pp. 246-261. Springer, Cham, 2017.

[8] Edling, Erik, and Emil Östergren. "An analysis of microservice frameworks." (2017).

[9] Seifermann, Valentin. "Application performance monitoring in microservice-based systems." Bachelor's thesis, 2017.

[10] Esposito, Christian, Aniello Castiglione, and Kim-Kwang Raymond Choo. "Challenges in delivering software in the cloud as microservices." IEEE Cloud Computing 3, no. 5 (2016): 10-14.

[11] Di Francesco, Paolo, Patricia Lago, and Ivano Malavolta. "Migrating towards microservice architectures: an industrial survey." In 2018 IEEE International Conference on Software Architecture (ICSA), pp. 29-2909. IEEE, 2018.

[12] Thönes, Johannes. "Microservices." IEEE software 32, no. 1 (2015): 116-116.

[13] Nikiforov, Roman. "Clustering-based Anomaly Detection for microservices." arXiv preprint arXiv:1810.02762 (2018).

[14] Zasadziński, Michał, Marc Solé, Alvaro Brandon, Victor Muntés-Mulero, and David Carrera. "Next stop" noops": Enabling cross-system diagnostics through graph-based composition of logs and metrics." In 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 212-222. IEEE, 2018.

[15] Zwietasch, Tim. "Online failure prediction for microservice architectures." Master's thesis, 2017.

[16] Thalheim, Jörg, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. "Sieve: Actionable insights from monitored metrics in microservices." arXiv preprint arXiv:1709.06686 (2017).

[17] Wu, Li, Johan Tordsson, Erik Elmroth, and Odej Kao. "Microrca: Root cause localization of performance issues in microservices." In NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1-9. IEEE, 2020.

[18] Eismann, Simon, Cor-Paul Bezemer, Weiyi Shang, Dušan Okanović, and André van Hoorn. "Microservices: A Performance Tester's Dream or Nightmare?." In Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 138-149. 2020.

[19] Mateus-Coelho, Nuno, Manuela Cruz-Cunha, and Luis Gonzaga Ferreira. "Security in Microservices Architectures." Procedia Computer Science 181 (2021): 1225-1236.

[20] Gkikopoulos, Panagiotis, Josef Spillner, and Valerio Schiavoni. "Monitoring Data Distribution and Exploitation in a Global-Scale Microservice Artefact Observatory." arXiv preprint arXiv:2006.01514 (2020).

[21] Knoche, Holger. "Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices." In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, pp. 121-124. 2016.

[22] Lavin, Alexander, and Subutai Ahmad. "Evaluating Real-Time anomaly detection algorithms--The Numenta anomaly benchmark." In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pp. 38-44. IEEE, 2015.

[23] Ueda, Takanori, Takuya Nakaike, and Moriyoshi Ohara. "Workload characterization for microservices." In 2016 IEEE international symposium on workload characterization (IISWC), pp. 1-10. IEEE, 2016.