International Journal of Advanced Technology and Engineering Exploration, Vol 5(43)
ISSN (Print): 2394-5443 ISSN (Online): 2394-7454
http://dx.doi.org/10.19101/IJATEE.2018.543011

Research Article

Automating live update for J2EE applications over distributed environment

Jalaj Pachouly*and Varsha Dange
Department of Computer Engineering, Dhole Patil College of Engineering Pune, India

Abstract
In the current world, there are very frequent changes in the industry needs due to pressing customer requirement, technology upgrade or fixing security issue in the currently deployed software, hence there is a great need of upgrading or updating the currently running applications. At the same time considering the usage of computers for almost in every domain, there is a lot of usage of horizontal scaling of software application where we are running multiple parallel servers deployed in various geographic locations across the globe using various cloud providers. It is desired to have automation for live update process. There is a great need to automate the process of upgrading the software version automatically, without manual intervention, without stopping the running servers, without losing the sanity of the application, smooth migration of coming request to newer version, no abrupt terminating the running process. In this paper it is recommended to use dynamic class loading with automated live update server, it will serve as a generic solution for live updating the multi-threaded servers. Using dynamic class loading, live update is not limited to just patching the server program with new content, instead we can update the business functionality. Dynamic class loading ensures that current running program did not get interfere while upgrade is taking place and after upgrade new version of the program takes over, and works smoothly for all new requests of computation. Hot deployment provides the smooth migration of software from old version to other, but still it is not fully automatic, across the location. The central live update server, which can perform upgrade on remote servers at multiple geographical location where the applications are deployed have been proposed.

Keywords
Terms-live update, DSU, Checkpoint-restart, Quiescence detection, Record-replay, Garbage collection, Threads.

1.Introduction
Coming up with a viable solution which can help organizations and mission critical applications to upgrade their deployed software's running in a distributed environment across geographical location, without manual intervention, using centralized control while keeping the sanity of the running system, without losing the integrity and data. As the prior work for live updating the deployed software is quite limited to apply security patches or hot deploying the partial functionality. In general upgrade includes the service restart with manual intervention is not suitable for many mission critical application like Medical and Military usage.

A. Objectives
The main objective of Live updating deployed software's is to keep downtime zero and make the process fully automatic, with smooth software upgrade in distributed environment, while preventing any data loss or bad user experience.

To achieve this requirement, it becomes quite obvious goal to ensure that on trigger of upgrade, event should reach to Remote server in time bounded manner; hence the primary internal goal is to attain the dynamic class loading and initialization of the new or upgraded service in an efficient manner. While doing live update of an application it is important to ensure correctness of the following aspects with a great quality like:

- Data integrity
- Smooth migration
- No data loss
- Fully Automated
- Centralized controlled
- Easy to trigger upgrade
- No customer interruption
- No service restart

B. Data integrity
Considering many threads running parallel on server, it is the desired that the live update is not impacting any running process and live data in erroneous way.

*Author for correspondence

C. Smooth migration
Live upgrade process should have good usability and easy to trigger and should finish quick and should clean the stale file or older contents of an upgraded application.

D. No data loss
It is very important to ensure that the data should not get lost while running live update and proper buffer or persistent state should be maintained for any important live data if required to ensure there is no data loss.

E. Fully automated
Once the user triggers the live update process from live update server, then there should not be any manual intervention on the application servers, and we should have the automation scripts which should unfold the upgraded contents and will do the needful for automatic upgrade process. It is important to note however that automation script should also be the part of the upgrade process.

F. Centralized controlled
Live update server will be centrally located and will be the place from where live update will be triggered to all registered application servers. There should be a user interface from where the user can monitor the live update process getting applied on the application servers.

G. Easy to trigger upgrade
Invoking live update should be as simple as clicking a button, in response to legacy mechanism where we need to manually copy the bundles of binary files and make them place at appropriate place, run couple of scripts and so on.

H. No Customer interruption
The whole idea is to make the upgrade frequent which will reduce time to market for new features in the product, hence it is important that upgrade process should not block any of the live customer operation and should be transparent to user. It should also get completed in expected time frame, which is in line with other servers, which are deployed at various geographic locations.

I. No service restart
There should not be any down time due to service restart for upgrading the software component so claim zero downtime for mission critical applications, as in case of Secure Live Update, code will be injected automatically by live update server.

2. Review of literature

There is a related work done on MCR, for server programs written in C language [1], using mutable check point restart algorithm. Another related but not similar technique "mutable record-replay" a previously executed and recorded execution of the same application to be replayed with altogether different version or upgraded version [2]. For supporting, arbitrarily complex software updates using generic C programs. Live update should support automatic state transfer and state validation [3]. Dynamic software updating (DSU) techniques helps to upgrade the critical business functionality, without stopping the server, and helps keeping no downtime, and hence protect data loss and state of the program[4]. Empirical testing about the effectiveness of DSU, and come up with conclusion that manual identification is most effective [5]. Automated solution is recommended hence, no manual programmer intervention and state of program is transferred automatically for program based on C language. It also uses very less program annotations [6, 7]. Usually a live update process is needed for a multithreaded server; hence needed dynamic software updates [8]. JVOLVE talks about coming up with a design of upgraded Java Virtual Machine. It is costly to stop and start the running servers for fixing critical bugs and security patches [9]. To fixes and updates to latest operating systems, it is required to have support for modular dynamic updates [10]. Comparison is shown in *Table 1*.

Table 1 Comparison with similar systems

Feature- Quiescence	Strengths	Weaknesses
Mixed Execution	No Quiescence needed	Annotation Required
Individual Function Quiescence	Easy Implementation	Inconsistent
Design Induced Quiescence	Native Support	Not for existing server
Explicit Per Thread Quiescence	Easy Implementation	No time bound
Profile Guided	Time Bounded	Profiling Required
My approach- Trigger Based	Time Bounded	No Profiling Required
Feature- State Transfer	Strengths	Weaknesses
In Place Updates	No Overhead	Developer to take care
Whole Program Update	No Restriction on change	Developer to take care
Type Transformers	Automatic Handle	Pointers not handled
My approach- Service Level	No Restriction	New Service Added

It's quite important to have an algorithm which can have a mechanism to do precise computations in distributed environment. Slice computing algorithm can be an option [11-15].

3.System architecture/system overview

Dynamic Class Loading is a proposed solution to achieve the mentioned goals here.

1) Using Dynamic Class Loading with supportive meta-data, developer requests a new version push.
2) Here developer's action triggers the upgrade process, reinitialized the server configuration to accept the re-quest changes.
3) It also transfers the state of current running programs to newer version.
4) The Live update server, with multiple sites running application which might need an upgrade for a newer version.
5) Live update communication will happen over a network link with proper Authentication.
6) For faster update, live update server use JAR's which a bundle of java classes is and other resources instead of sending upgraded file individually.

Live update server configuration is shown in *Figure 1*.

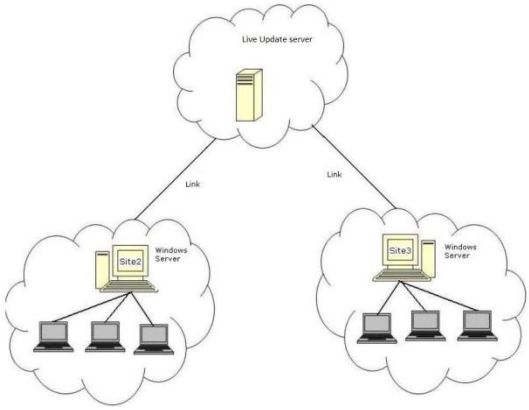


Figure 1 Live update server configuration

Below are the few important modules used in the live update system.

A. Dynamic class loading

Dynamic class loading in Java is a feature using which we can load or reload a class dynamically. As the same class loader which had loaded the class initially cannot load the class again, hence we need to use Custom class loader to load the new class. although it is easy to load a class using custom class loader, but it also poses some challenges, like same class loaded by class loader A will not be same as loaded by class loader B, and we can't type cast the two classes. if we can resolve the issues of relinking the classes then it can be an alternative to main content repository MCR mechanism, where we have to wait for a certain time to figure out the proper time frame where all threads are in a waiting mode where we can reinitialize the server main configuration process and get the proper run time environment with the updated code. In the current implementation, we have used the dynamic class loading feature with java reflection to load the new and updated services at run time. Reflection helps to get the properties of a new class, the operations supported and the class attributes. Using reflection we can also invoke the new methods which will provide the new behavior to the old program, in essence executing the updated method instead of the old one. We will also need a supportive meta-data which we can use to represent the new state of the program any server where we want a hot deployment facility should support this meta-data, using which we can smoothly perform the upgrade.

B. Live update server

Live update server is a central server, which hosts the latest upgrade in the software versions, and pushes those upgrades to various sites and machines on a single click. Live update is shown in *Figure 2*.

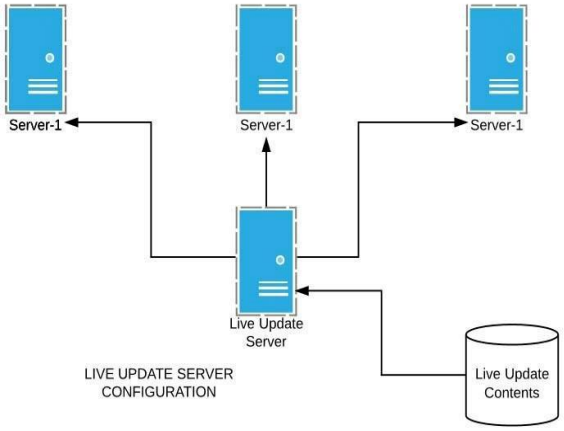


Figure 2 Live update

C. Authentication manager

Authentication manager ensures that live update server and the various nodes in the environment recognize each other, and should not accept any unauthenticated communication for server upgrades. It is important to perform the threat modelling to ensure that server are not injected with malicious code while upgrade by intruders, hence authentication should be in place.

D. JAVA archive

JAR is Java archive which contains all the required artifacts which are required to fulfil the business functionality and usually need to be in the class path of the deployed application which are running on a remote server. So to get the new feature on the remote server, it is required to bundle all the modified files together in a JAR file and send them to the remote server over the wire. There must be a process on the remote server which unfold this bundle and place the resources in the class path according to their classification like class file should go to BIN folder, JSP and images should go the WEB folder etc.

4.System analysis

System Analysis describes the mathematical model, high level algorithm steps and various measures used to analyze the proposed system. Checking the efficiency of the system to ensure the proposed solution is feasible and practical in business usage.

Figure 3 shows the Venn diagram depicting software upgrade.

A. Mathematical model

$S = \{S1, S2, S3 \ldots . Sn\}$

Here S is set of Servers running in parallel

$L \rightarrow Centrally\ Located\ Live\ Update\ Server$

$T = \{T1, T2, T3 \ldots . Tn\}$

Here T is set of trigger events generated by L

$Q = \{Q1, Q2, Q3 \ldots . Qn\}$

Here Q is a set of Quiescence state for various servers

Ideal case

$Q1 \times t = Q2 \times t = Q3 \times t = \cdots\ Qn \times t$

Here t is a time to reach the Quiescence state

$R = \{R1, R2, R3 \ldots . Rn\}$

Here R is a set of re-initialization time of server after upgrade.

$R1 = R2 = R3.. \ = Rn$

$Input \rightarrow Update\ Trigger\ Event\ to\ Dynamic\ Class\ Loading\ enabled\ server = T$

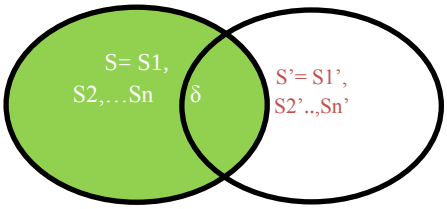
$Output$

$\rightarrow Server\ reinitialization\ after\ \ upgrade$

$S' = \{S'1, S'2, S'3 \ldots . S'n\}$

$S = S + \delta$

$\delta \rightarrow Change\ in\ Software$



$S' = S + \delta$

$\delta \rightarrow$ Change in Software

Figure 3 Venn diagram depicting software upgrade

B. Algorithm

1) Load new version of the software in the central live update server.
2) Configure Remote server with Live Update server.
3) Initiate the trigger event on Remote Server for upgrade.
4) Unzip the archive in remote machine.
5) Copy the resources to the class path of the running server on remote machine from Jar.
6) Dynamic load the new/upgraded service in Remote machine using REST enabled API.
7) Send Success message to Live Update server if successful else send error message.
8) Service the request using new/upgraded service in Live Update Server.

C. Performance measures used

Following are the important performance measures which can be used to calculate the efficiency and effectiveness of the proposed system.

- Memory Requirement for upgrade execution
- CPU usage for the upgrade
- Time to upload the new version of the Software at Remote
- Server Triggering Upgrade and Initializing new version of the service
- Total turnaround time for Upgrade Process of N number of remote servers

D. Efficiency calculation

Efficiency of the Centralized Live Update Server can be calculated in two aspects:

1) Memory efficiency: Need of extra memory in upgrade process.

Percentage memory efficiency =

$$\left(\frac{\text{Memory requirement before upgrade}}{\text{Total memory after upgrade}}\right) \times 100$$

2) Time efficiency: Time to upgrade n number of server from one version to another.

Percentage time efficiency =

$$\left(\frac{\text{Time to manually upgrade}}{\text{Time to upgrade with live upgrade server}}\right) \times 100$$

5.Implementation

Implemented live update solution for Java/J2ee application deployed on Tomcat Apache server, Implementation is done using REST full API's, Spring Boot, Dynamic Class Loading, Mongo DB, HTML5, Angular JS open source framework. Couple of modules like:

- User module
- Configuring remote server on live update server
- Session management
- Displaying summary for users and configured servers
- Uploading new version in live update server
- Trigger upgrade event
- Auditing important events like log in, trigger and upload actions

6.Results

Centralized Live update can invoke live update on N numbers of servers. By general observation it is seen that upgrade tasks completes for medium size of an upgrade (below 500 KB) in approximately 2 minute per remote server, whereas manual hot deployment for the same takes near about 10 minutes per server, due to manual intervention, hence time efficiency of the system can be approximately given as,

Percentage time efficiency =

$$\left(\frac{\text{Time to manually upgrade}}{\text{Time to upgrade with live upgrade server}}\right) \times 100$$

$$= \left(\frac{600}{120}\right) \times 100$$

$$= 500\ \%$$

A. CPU Usage

Below is the measurement taken for 10 readings for the usage of the CPU for trigger live update action. It is on central live update server and the remotely deployed application server, which we need to upgrade (*Table 2*). We can see that live update is quite fast, and time taken for 67KB payload is very less, few milliseconds only.

Table 2 CPU usage for the 10 reading-action-"triggers live update on remote server"

Reading No	LU server(ms)	Remote server(ms)
1	19	476
2	18.4	471
3	39.6	476
4	18.7	479
5	22.1	469
6	265.4	476
7	29.2	476
8	27.5	472
9	17.6	481
10	21	475

Based on the above mentioned *Table 2*, we can quickly observe that initiating the trigger action can be quite quick from the central live update server; hence we can trigger live update operation in bulk in many remote application servers. It takes little more time to perform the actual live update on remote application server in respect to the trigger action, although it will still remain constant as it is executed in separate path of execution on individual remote server, hence the time to deploy the latest upgrade almost remains constants. *Figure 4* shows the time in millisecond on central live update server versus remote application server. *Figure 5* shows the centralized versus manual live update result comparison. *Figure 6* shows the centralized live update resources. *Figure 7* shows the remote application server resources.

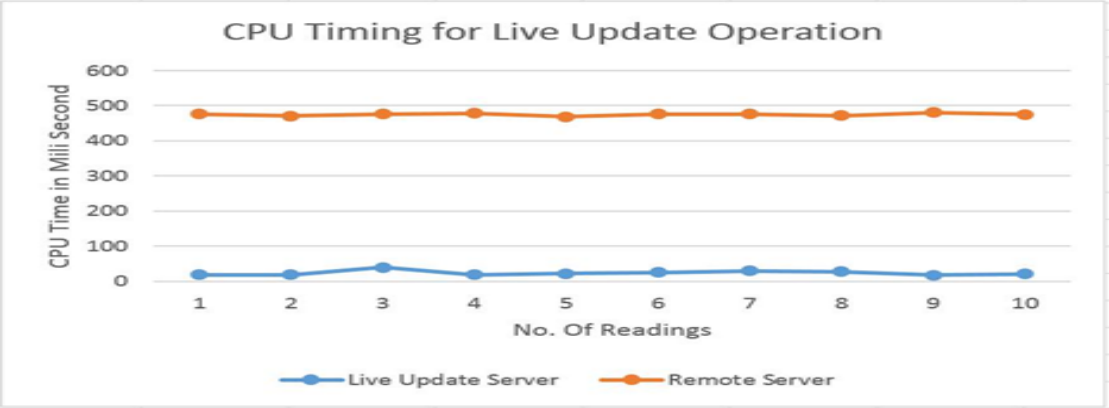


Figure 4 Time in millisecond on central live update server versus remote application server

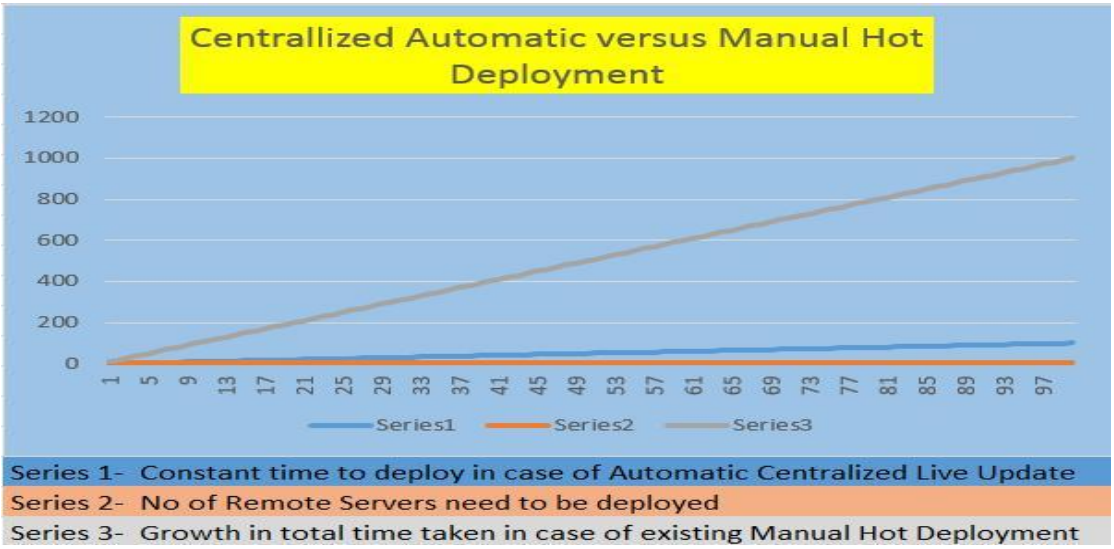


Figure 5 Centralized versus manual live update result comparison

B. System Resources Monitoring on Live Update Server

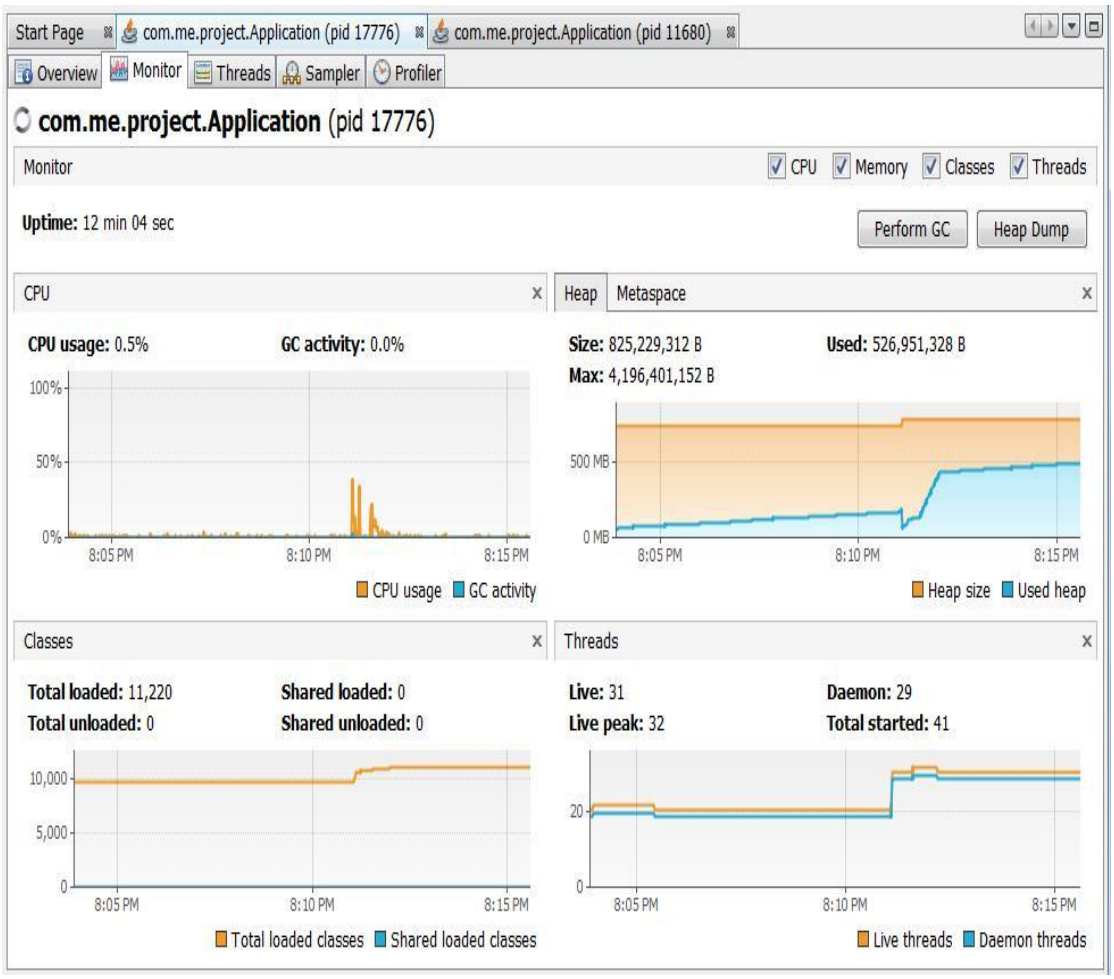


Figure 6 Centralized live update resources

C. System Resources Monitoring on Remote application Server

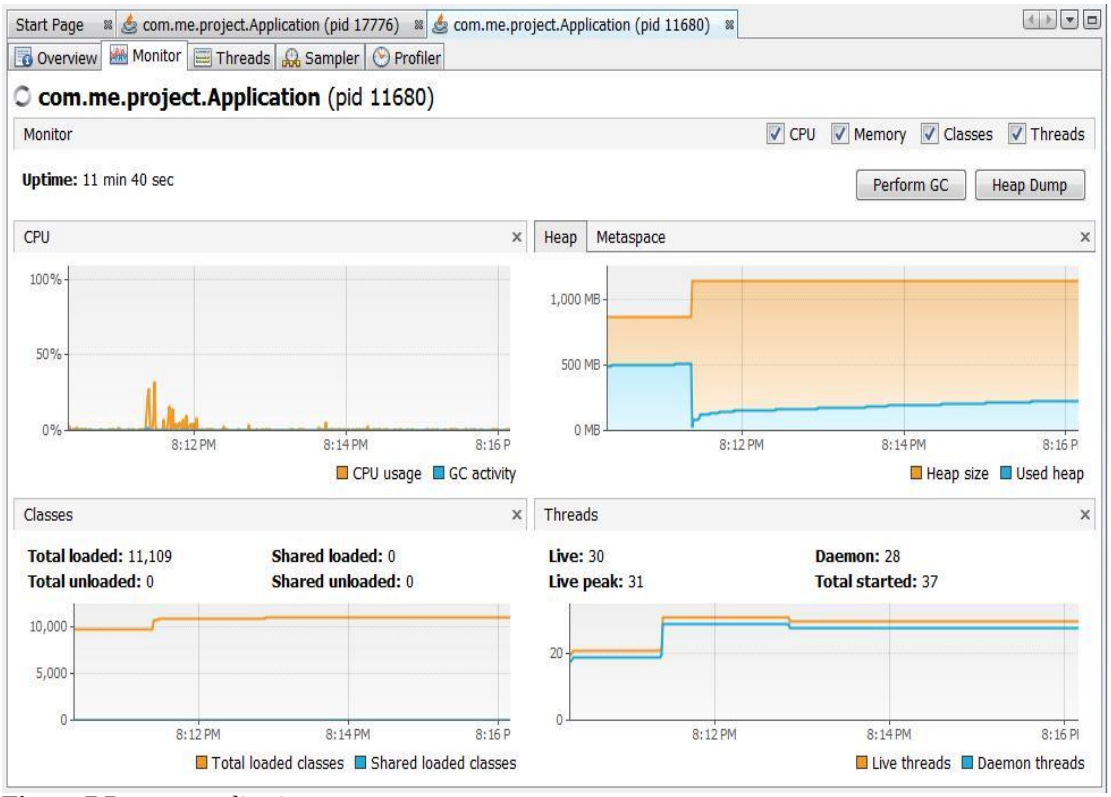


Figure 7 Remote application server resources

7.Conclusion

Using Centralized Live Update server, we can keep down-time zero and make the upgrade process fully automatic, with smooth software upgrade in distributed environment, while preventing any data loss or bad user experience. As we have seen that Dynamic Class Loading from java run time environment can be very well leveraged to achieve the below mentioned objectives:

1. Fully automated software upgrade.
2. Cost saving as no manual intervention needed.
3. Centralized control over upgrade process.
4. Smooth migration of running process to higher version.
5. Maintaining sanity and secure upgrade

In current research attempt, we have focused on the no SQL database and JSON communication across distributed components, for J2ee server which are very generic in nature and platform independent, in future we should approach Live Update for legacy applications using Relation Database and have heterogeneous platform level dependency.

AS of now we need Live update admin, in future we can also automate the live update trigger automatically the moment we received new software version.

Acknowledgment

I would like to thank Prof. Varsha Dange, Head of Department (Computer Engineering), Dhole Patil college of Engineering Wagholi Pune, for providing guidance on various aspects, specially analysis and reporting of the contents, coming of the overall architecture and design, finalizing the scope of analysis and helping to come up with the Dissertation report and project work. I also would like to thank Prof. Vandana Navale, (ME Coordinator), Dhole Patil college of Engineering Wagholi Pune, who guided and encouraged me in completing various tasks related to project, and provided suggestions to improve understanding of the various topics.

Conflicts of interest

The authors have no conflicts of interest to declare.

References

[1] Giuffrida C, Tamburrelli G, Tanenbaum AS. Automating live update for generic server programs. IEEE Transactions on Software Engineering. 2017; 43(3):207-25.

[2] Viennot N, Nair S, Nieh J. Transparent mutable replay for multicore debugging and patch validation. In ACM SIGARCH computer architecture news 2013 (pp. 127-38). ACM.

[3] Giuffrida C, Iorgulescu C, Kuijsten A, Tanenbaum AS. Back to the future: fault-tolerant live update with time-traveling state transfer. In LISA 2013 (pp. 89-104).

[4] Hayden CM, Saur K, Hicks M, Foster JS. A study of dynamic software update quiescence for multithreaded programs. In workshop on hot topics in software upgrades 2012 (pp. 6-10). IEEE.

[5] Hayden CM, Smith EK, Hardisty EA, Hicks M, Foster JS. Evaluating dynamic software update safety using systematic testing. IEEE Transactions on Software Engineering. 2012; 38(6):1340-54.

[6] Giuffrida C, Tanenbaum AS. Safe and automated state transfer for secure and reliable live update. In proceedings of the international workshop on hot topics in software upgrades 2012 (pp. 16-20). IEEE Press.

[7] Kolbitsch C, Kirda E, Kruegel C. The power of procrastination: detection and mitigation of execution-stalling malicious code. In proceedings of the ACM conference on computer and communications security 2011 (pp. 285-96). ACM.

[8] Makris K, Bazzi RA. Immediate multi-threaded dynamic software updates using stack reconstruction. In USENIX annual technical conference 2009.

[9] Subramanian S, Hicks M, McKinley KS. Dynamic software updates: a VM-centric approach 2009(pp.1-12). ACM.

[10] Baumann A, Appavoo J, Wisniewski RW, Silva DD, Krieger O, Heiser G. Reboots are for hardware: challenges and solutions to updating an operating system on the fly. In proceedings of the USENIX annual technical conference 2007.USENIX Association.

[11] Johnson P, Mittal N. A distributed termination detection algorithm for dynamic asynchronous systems. In IEEE international conference on distributed computing systems 2009 (pp. 343-51). IEEE.

[12] Subhraveti D, Nieh J. Record and transplay: partial checkpointing for replay debugging across heterogeneous systems. In proceedings of the joint international conference on measurement and modeling of computer systems 2011 (pp. 109-20). ACM.

[13] Laadan O, Viennot N, Nieh J. Transparent, lightweight application execution replay on commodity multiprocessor operating systems. In ACM SIGMETRICS performance evaluation review 2010 (pp. 155-66). ACM.

[14] Altekar G, Stoica I. ODR: output-deterministic replay for multicore debugging. In proceedings of the symposium on operating systems principles 2009 (pp. 193-206). ACM.

[15] Moseley T, Grunwald D, Connors DA, Ramanujam R, Tovinkere V, Peri R. Loopprof: dynamic techniques for loop detection and profiling. In proceedings of the workshop on binary instrumentation and applications 2006 (WBIA).

Jalaj Pachouly is pursuing Master's degree in Computer Engineering from Dhole Patil College of Engineering Pune. His area of interest are Cyber Security, Software architecture and Engineering Design and Cloud computing.
Email: jalajpachouly@gmail.com

Prof. Varsha Dange having 12 years of experience in teaching, currently working as Head of the department, Computer Engineering in Dhole Patil College of Engineering. Her area of interest are Database Management Systems, Network Security, Cloud Computing, Parallel Computing.