LIVE UPDATING J2EE APPLICATIONS DEPLOYED ON CLOUD OR DISTRIBUTED ENVIRONMENT

Jalaj Pachouly[1], Prof. Varsha Dange[2]
[1]Student, Department of Computer Science, Dhole Patil College of Engineering Pune
[2]Professor, Department of Computer Science, Dhole Patil College of Engineering Pune

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 server deployed in various geographic locations across the globe using various cloud providers. As mentioned , 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,Not abruptly terminating the running process.Current proposal recommends to use Mutable Checkpoint-Restart (MCR), a new live update solution for generic (Multi-process and Multi-threaded) server programs written in C. MCR can support arbitrary software updates and automate most of the common live update operations, by allowing the running version to safely reach a quiescent state and then allow the new version to restart as similarly to a fresh program initialization as possible.MCR provides the smooth migration of software from one version to other, but still it is not fully automatic, across the location. I am proposing the central live update server, which will be connected to multiple geographical location where the application is deployed, and will automatically push the upgraded version of the software with a click of a button. At the same time, it will also perform authentication before upgrading the version to avoid any malicious at temp to upgrade the software from unknown servers or third party with bad intentions.

Index Terms: Live update, DSU, checkpoint-restart, quiescence detection, record-replay, garbage collection, Threads, OSGI.

Introduction

Coming up with a viable solution which can help organizations and mission critical applications to upgrade there 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 loosing 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 which includes the service restart with manual intervention is not suitable for many mission critical application like Medical and Military usage. Hence it is desired to come up with some system which provides the full program upgrade without downtime of the server while maintaining the required quality parameters like.

- Data Integrity

  Considering many threads running on server, it is the desired quality that live update is not

impacting any running process and live data in erroneous way.

- 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.
- 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.
- 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.
- 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.
- 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. • 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.
- 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.

The main goal 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 System should reach to Quiescence state in time bounded manner, hence the primary internal goal is to reach the Quiescence state using efficient MCR algorithm.

To summarize

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

Proposed Live Update System

- Propose profile-guided quiescence, a technique which allows all the program threads to automatically and safely block in a known quiescent state using dedicated information gathered during an offline profiling phase.
- Propose mutable re initialization, a technique which record-replays start up operations between different program versions and exploits existing code paths to automatically reinitialize the new program version, its threads, and a relevant portion of its global data structures.
- Propose mutable tracing, a technique which transfers the remaining data structures between versions using precise (when possible) and conservative (otherwise) GC-style tracing strategies. Benchmark the e effectiveness of our

techniques in Mutable Checkpoint-Restart, a new live update solution for generic server programs written in C/Java.

- Providing hot deployment with Completely automated process which is driven by central Live update server, targetting Java/J2ee servers and programs.

Relevant mathematics associated with the Project

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

Here S is a set of Servers running in parallel

$L \rightarrow$ Centrally Located Live Update Server

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

Here T is a set of Trigger events generated by L

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

Here Q is a set of Quiescence state for various servers

Ideal Case :

$Q1t = Q2t = Q3t:::: = Qnt$

Here t is a time to reach the Quiescence state

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

Here R is a set of Reinitialzization time of server after upgrade

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

Input → Update Trigger Event to MCR enabled server = T

Output → Server re-initialization after upgrade

$S^0 = \{S1^0, S2^0, S3^0, ..Sn^0\}$

$S^0 = S + \delta$

$\delta \rightarrow$ Change in software

MCR Enabled Server Program

Listing 1. A sample MCR-enabled server program.

```
1   /* Auxiliary data structures. */
2   char b[8];
3   typedef struct list_s {
4     int value;
5     struct list_s *next;
6   } l_t; l_t list;
7
8   /* Startup configuration. */
9   struct conf_s *conf;
10
11  /* Server implementation. */
12  int main() {
13    server_init(&conf);      // startup
14    while(1) {               // main loop
15      void *e = server_get_event();
16      server_handle_event(e, conf, b, &list);
17    }
18    return 0;
19  }
20
21  /* MCR annotations. */
22  MCR_ADD_OBJ_HANDLER(b, custom_b_handler);
23  MCR_ADD_REINIT_HANDLER(custom_reinit_handler);
24  MCR_ADD_QUIESC_HANDLER(custom_quiesc_handler);
```

Fig 1: MCR Enabled Server[1]

```
1: procedure COORDINATOR
2:     Q ← 1
3:     repeat
4:         A ← 0
5:         SYNCHRONIZE_RCU()
6:         SYNCHRONIZE_RCU()
7:     until A = 0
8:     Q ← 2
9:     SYNCHRONIZE_RCU()
10:    SYNCHRONIZE_RCU()
```

```
1: procedure QUIESCENTPOINT
2:     if Q > 0 then
3:         if Active then
4:             A ← 1
5:         if Initiator or Q > 1 then
6:             RCU_THREAD_OFFLINE()
7:             THREAD_BLOCK()
8:             RCU_THREAD_ONLINE()
9:             THREAD_UNBLOCKED()
10:    RCU_QUIESCENT_STATE()
```

Fig 2: Quiescence detection protocol pseudocode.[1]

The COORDINATOR code runs on a separate thread. The QUIESCENTPOINT is called by application threads when a quiescent point is reached.

Methodology and Architecture

Mutable Checkpoint-Restart (MCR) is a proposed solution to achieve the mentioned goals here.

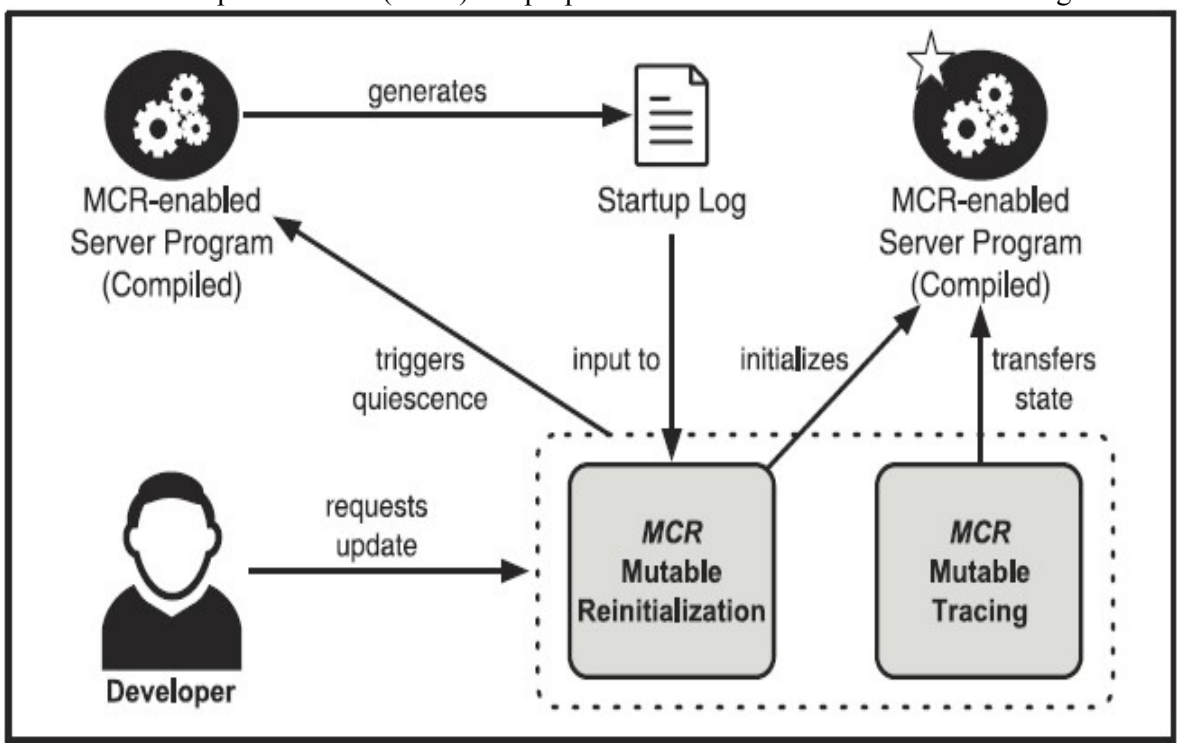

Fig 3: System Architecture[1]

Fig. 3 illustrates

1. The MCR process where we have MCR enabled server, and developer request a new version push.
2. Here developers action triggers the quiescence state in the MCR enabled server, reinitialized the server configuration to accept the request changes.
3. It also transfer the state of current running programs to newer version .

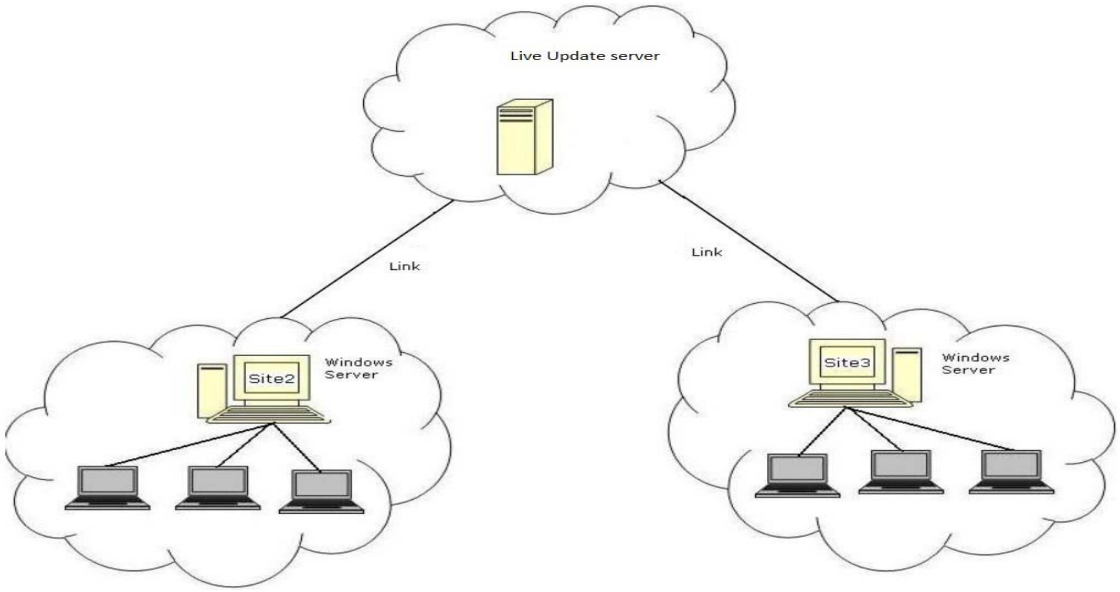

Fig 4: Live Update Process

Fig. 4 illustrates

1. The Live update server , with multiple sites running application which might need an upgrade for a newer version.
2. Live update communication will happen over a network link with proper authentication
3. for faster update, live update server compress the binary files which will get uncompressed over the sites and will be deployed.

Modules Details

1. MCR.

Checkpoint-Restart (MCR), a new live update solution for generic (multiprocessing and multithreaded) server programs written in /java. MCR can support arbitrary software updates and automate most of the common live update operations. The key idea is to allow the running version to safely reach a quiescent state and then allow the new version to restart as similarly to a fresh program initialization as possible, relying on existing code paths to automatically restore the old program threads and reinitialize a relevant portion of the program data structures. To transfer the remaining data structures,.

As mentioned it is very important that all threads running in an application are reached to quiescent state. then only we apply the upgrade process.In fact when we start the live update process, then there will be a mechanism which is based on profile guided updates and will lets threads to keep executing until they reach to a safe stable state and then make them wait unless the live update process doesn't get complete,the same is true for all running threads in the system. MCR point can purposefully introduced in the code, or can be decided by profiling the application.Once the live update is triggered, system should not wait indefinitely to reach the quiescent state. It is important to ensure that no additional request should be accepted by the system, once the live update process started on the targeted server, and the only ask should be reach to quiescent state and make the upgrade success full. This process need to be efficient, as we might be pausing the server request handling when switching from one version to another version, the whole purpose of this quick transition to avoid impacting customers request and any kind of downtime experience for the end users.

2. Live Update Server. Live update server is a central server, which hosts the latest upgrade in the software versions, and push those upgrades to various sites and machines on a single click. Server.png

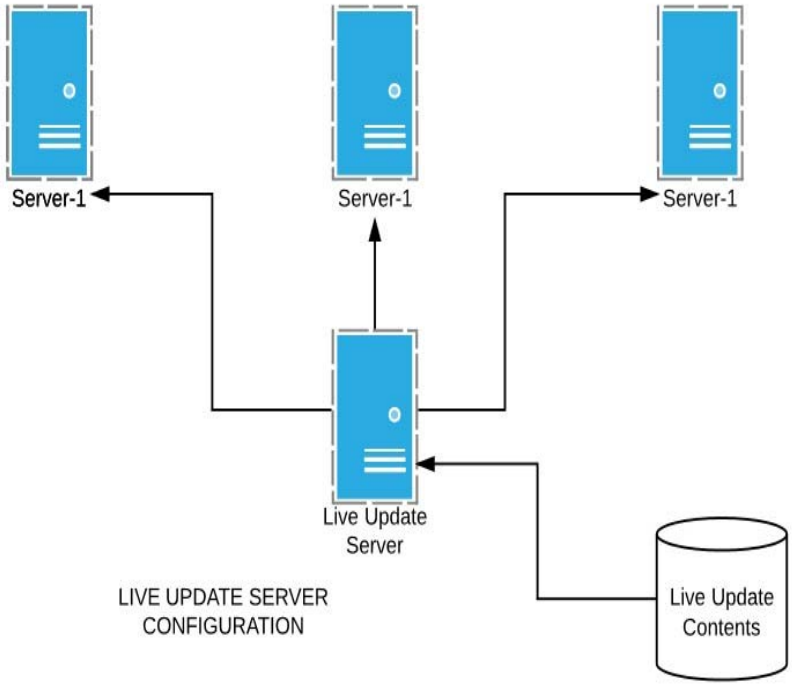

Fig 4: Live Update Process

3. 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 and upgrade process is using secure connection.

4. File Compressor. This is the module which will compress the version upgrade binaries to small size archive file to faster download. The compressed file will get un-compressed once downloaded and triggers the MCR process on the server. Compression will ensure that network bandwidth is not wasted and it will take less time to move upgraded content from live updated server to actual server where application is running. It is recommended to use some frameworks like OSGI which contains smart jars which has self initialization capacity hence we can bundle only relevant modules and upgrade the system with minimal content transfer. This will increase the upgrade process immensely and many servers will get upgraded quickly.

References

1. Cristiano Giuffrida, Member, IEEE, Clin Iorgulescu, Giordano Tamburrelli, and Andrew S. Tanenbaum, Fellow, IEEE, Automating Live Update for Generic Server Programs,IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 43, NO. 3, MARCH 2017.

2. N. Viennot, S. Nair, and J. Nieh, Transparent mutable replay for multicore debugging and patch validation, in Proc. 18th Int. Conf. Archit. Support Program. Lang. Oper. Syst., 2013, pp. 127138.

3. A. Kuijsten, Cristiano Giuffrida, Calin Iorgulescu, and A. S. Tanenbaum, Back to the future: Faulttolerant live update with time-traveling state transfer, in Proc. 27th Int. Conf. Large Install. Syst. Adm., 2013, pp. 89104.

4. C. Hayden, K. Saur, M. Hicks, and J. Foster, A study of dynamic software update quiescence for multithreaded programs, in Proc. 4th Int. Workshop Hot Top. Softw. Upgrades, 2012, pp. 610.

5. C. Hayden, E. Smith, E. Hardisty, M. Hicks, and J. Foster, Evaluating dynamic software update safety using systematic testing, IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 13401354, Nov./Dec. 2012. [30] C. Giuffrida, C.
6. C. Giuffrida and A. Tanenbaum, Safe and automated state transfer for secure and reliable live update, in Proc. 4th Workshop Hot Top. Softw. Upgrades, pp. 1620. Jun. 2012
7. C. Kolbitsch, E. Kirda, and C. Kruegel, The power of procrastination: Detection and mitigation of execution-stalling malicious code, in Proc. 18thACMConf. Comput. Commun. Sec., 2011, pp. 285296.
8. K. Makris and R. Bazzi, Immediate multi-threaded dynamic software updates using stack reconstruction, in Proc. Conf. USENIX Annu. Tech. Conf., 2009, pp. 3131.
9. S. Subramanian, M. Hicks, and K. S. McKinley, Dynamic software updates: A VM-centric approach, in Proc. 30th ACM SIGPLAN Conf. Program. Lang. Des. Implementation, 2009, pp. 112
10. A. Baumann, J. Appavoo, R. W. Wisniewski, D. D. Silva, O. Krieger, and G. Heiser, Reboots are for hardware: Challenges and solutions to updating an operating system on the fly, in Proc. USENIX Annu. Tech. Conf., 2007, Art. no. 26.