ARTICLE

AN EXTENSIBLE FRAMEWORK USING MOBILITYRPC FOR POSSIBLE DEPLOYMENT OF ACTIVE STORAGE ON TRADITIONAL STORAGE ARCHITECTURE

Prashant Desai[*], **Naveenkumar Jayakumar**

Department of Computer Engineering, Bharati Vidyapeeth (Deemed to be University), College of Engineering, Pune, INDIA

ISSN 0976-3104

ABSTRACT

Background: *Nearly 2.5 quintillion bytes of data is generated every day and will increase ten folds by the year 2020 when Internet of Things (IoT) takes global domination in today's digital universe. Ninety percent of data generated is of "unstructured type" consisting of videos, audios, logs and sensor data, which are difficult to process. Because of the large amount of data generated and I/O intensive task being created, data transfer rate are proving to be bottlenecks in performance.* ***Methods:*** *In this research paper, we propose an active storage architecture using MobilityRPC for offloading code near to the data logic.* ***Results:*** *Both the traditional and proposed system is tested against the same synthetic workload and a reduction of 35.36 % in execution time on the proposed system is achieved.* ***Conclusions:*** *To conclude its fair say that the proposed active storage model works very well for data intensive applications where I/O bound scans are more and offloading the application is rather beneficial than the traditional method.*

.

KEY WORDS
Active storage, MobilityRPC, Application offloading, data intensive

INTRODUCTION

With the proliferation in processor speed and storage capacity, we assess Active Storage architecture as an alternative that exploits processing capabilities embedded in a disk drive to execute application. Moreover, see it has a viable and a profitable option to move computation closer to where data resides. Active Disks, which are based on the same ideology as that of Active Storage architecture of offloading computation to disk drives comes with numerous advantages such as less I/O bound scans, a higher degree of parallelism and I/O interconnect because of each disk having its own processor [1]. However, there are certain disadvantages such as system with large configuration; the disk can communicate only through the host interface leading to bottlenecks and its inability to process complex applications. Intelligent Disks (IDisks) can be seen as the successors of Active Storage with the ability to process complex applications at the cost of higher hardware resources [2]. Hadoop is a relatively new storage paradigm, which embraces the concept of Active Storage. It offers ample of advantages such as scale-out architecture, reliability, capability to store huge volumes of data and processing logic closer to data rather than the traditional way.

One of the principal techniques employed by the architectures mentioned above is offloading or migrating of application code using techniques such as code mobility, code on demand, remote evaluation and mobile agents to execute an application on the storage device. So using a mobility framework, we propose an architecture that encompasses active storage principals. The rest of the paper is divided into the following sections; a survey of Active Storage concepts and technologies, architecture of the proposed system and the technologies used implementation of the proposed system and a comparative result analysis with a traditional system.

Received: 24 April 2018
Accepted: 20 May 2018
Published: 2 June 2018

LITREATURE REVIEW

Acharya *et al.*propose a stream-based programming model that is designed to take advantage of the Active Disk architecture. Active Disk function by partitioning an application into host-occupant and disk-occupant components. The data-intensive tasks of the applications are offloaded to the disk- resident processors, while the entire task is managed and coordinated by the host-resident processors. The proposed stream-based model is for the disk-resident code also known as Disklets [1] and its communication with the host. A Disklet is language independent, take data stream as input, and produce data streams as output. It is designed to run in a sandbox with restricted privileges such as, cannot allocate memory and initiate I/O calls with the system. The Active Disks require the support of operating system at both, system and disk level, the proposed stream-based model takes advantage by simplifying resource allocation; it assigns memory in contagious blocks and provides stream buffers for communication [1]. A number of algorithms are proposed to compare and benchmark the performance of Active disk to conventional disks.

Keeton *et al.* propose an architecture for decision support systems (DSS) in which the servers use disk embedded with the microprocessor for offloading computation from the host processor. The disks are known as IDISKS [2] and are interconnected by high-speed crossbar switches for communication. It offers several advantages in terms of cost and performance over traditional server architecture. By embedding processor on the disks, computation is moved near data reducing data movement through the I/O system. The disks are interconnected to a high-speed switch thus eliminating the I/O bottleneck of the conventional disk and allowing storage devices to truly scale with data growth.

***Corresponding Author**
Email: prashantdesai99@gmail.com
Tel.: +91- 8806507647

IIOAB JOURNAL ISSN: 0976-3104

Anastasiadis *et al.*propose Lerna [3] an Active storage framework for executing application close to the data. The goal is to create an execution environment within the network file server that will allow the users and administrators to run executable code. In the architecture, multiple storage nodes in a server are arranged into multi-tier hierarchy assembly. The client mounts the servers as normal remote file servers, and the files and directories can be accessed with transparency. The proposed architecture uses URL encoding scheme to convert command line statements into file name strings [3]. The remote services can receive data from client machines by remote method invocation.

Ma *et al.*propose a multiview storage system (MVSS) framework [4] for offloading services to storages devices. The concept of MVSS views is similar to database views; they are generated dynamically and not stored on the storage devices. These views are presented to the users through the file system namespace. It uses block interface that supports a wide range of platforms allowing the reuse of existing file system. It introduces the concept of virtual files; it is the combination of files and certain services. Virtual disks are an abstract version of physical drives and contain the virtual files. In MVSS, new services can be dynamically created from existing ones by downloading apiece of code onto the device [4]. To add write support and cache coherency it MVSS implement a lightweight file access protocol, which is not efficient to previous implementations but simple.

Riedel *et al.*propose a system called Active Disks for exploiting the processing power of disks to run application code. They proposereal-time applications such as nearest neighbor search, data mining and edge detection [5] for benchmarking the Active Disks. The basic approach is to utilize to thedegree of parallelism offered by Active Disks, operate with minimal state information, process data while streaming from disks directly and execute arelatively small number of instructions per byte. The speedup offered by the proposed analytical model is twice of that of conventional disk drives. For sorting algorithms, data is read into the nodes from the local drives exchanged across the network according to key space distribution and sorted on the local nodes independently.

Joo *et al.*propose a virtual machine based platform IOLab, [6] for executing intelligent functions efficiently on active storage devices. It has negligible performance overhead and provides theuser with latest intelligent functions.IOLab is a user space module that is interpolated between a virtual machine and the virtual file system (VFS) of a host. IOLab intercepts I/O requests from the VM and forwards it to the intelligent functions executing in the IOLab. IOLab is implemented, as an application on the host hence does not reply on dedicated driver support. It implements the concept of active storage by running commodity block devices. The block cache stores the data blocks requested by the VM and are serviced by IOLab instead of being redirected directly to active storage. The I/O dispatcher performs operations such as data replication, migration, and prefetching in the IOLab. By defining the maximum block cache size the dispatcher rate can be varied.

Runde *et al.* propose a framework for the execution of object storage devices downloaded from the host. The framework is based on the use of virtual machines for providing processing capabilities and executing the offloaded code. The framework is built upon an open source OSD stack, with the core of the framework running the desired virtual machine [7]. The current version of the platform has a C API engine running on a Linux OSD. The VMs are sandboxed to prevent unauthorized execution of code snippets. RPC model is the central programming model for this framework. In order to implement commands on active storage remotely the OSD functions are modified with execute_function command that executes functions that exist as objects on the storage [7]. The proposed platform provides support for multiple engines allowing simultaneous execution of functions and offers high degree of parallelism.

Yin *et al.*propose architecture for active storage systems and a hybrid disk configuration for the storage system in active storage. Termed as the HcDD the disk architecture consist of dual buffer; [8] a HDD write buffer for deduplication of write requests and SSD buffer for providing parallelism to write requests. Read requests are directly processed by the SSD whereas write request to an active storage system is sent to the HDD buffer for deduplication before processing. The major contribution of this research is to reduce the number of write requests sent to the SSDs thereby reducing the number of erasure cycles, and therefore increasing the lifespan of SSDs. The proposed model integrates HDDs and SDDs to create hybrid disk architecture, before the write requests are written they deduplicated in the active storage. The processor on the disk calculates the hash value for each page in the buffer. The hash value is used as a fingerprint to lookup data on the SSD.

PROPOSED SYSTEM ARCHITECTURE

The proposed system architecture for an active storage is given below in [Fig. 1]. We use the following technologies for implementing the proposed system.

Virtualization: The data node is virtualized using a bare metal hypervisor [9] also known as the type-1 hypervisor, this type of virtualization provides complete control over the hardware, provides better scalability and performance compared to hosted hypervisors [9]. It also has the ability the host operating systems. VMware ESXi is used to host an operating system. The application migrated from the application node is offloaded and executed on this hosted operating system in contrast to the traditional system which involves moving data across the network on demand. VMware ESXi provides a host of tools for monitoring and recording the performance of operating systems.

ISSN: 0976-3104

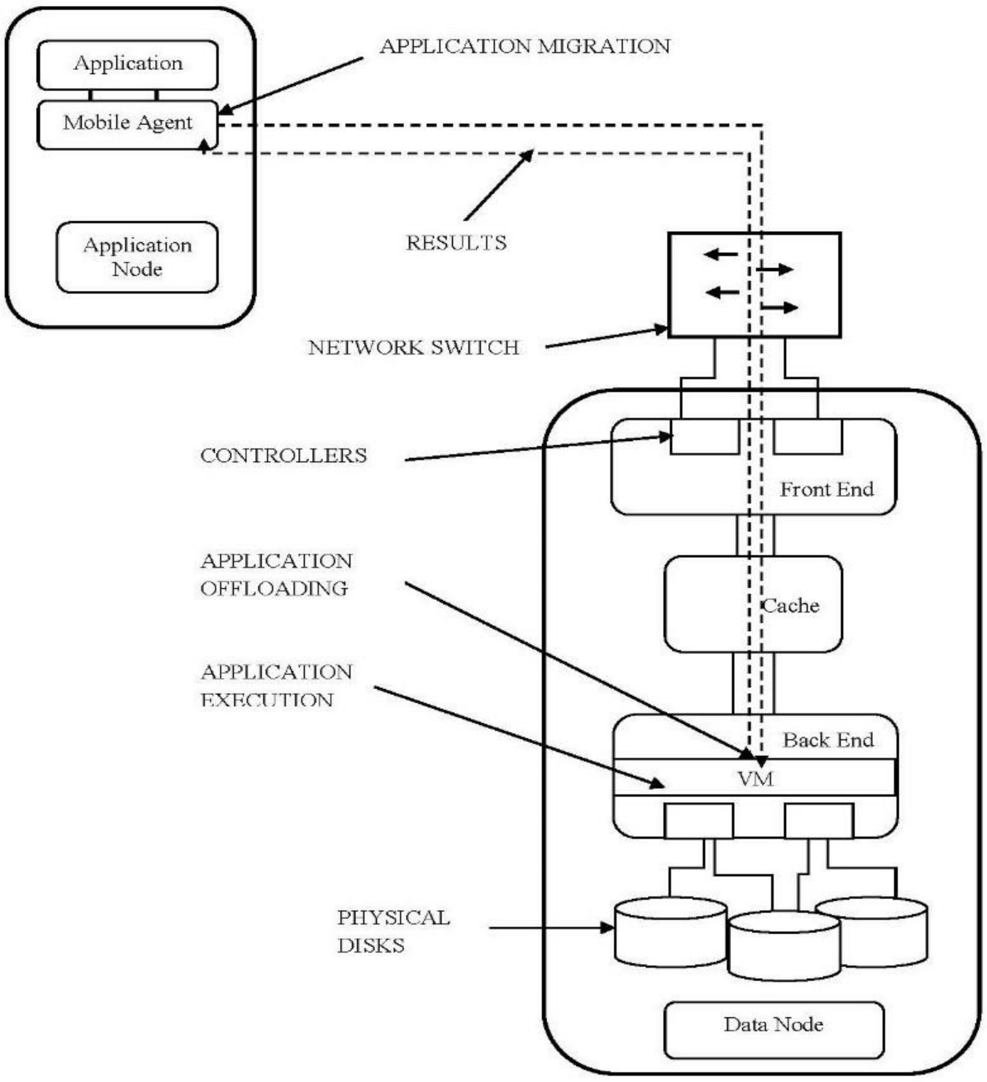

Fig. 1: Proposed system architecture.

MobilityRPC: It is a mobile agent framework used for migrating code between two nodes. It has the ability to write objects, which can move within a network and invoke arbitrary methods remotely [10]. It facilitates communication between two machines by defining a protocol, which allows class loaders to communicate and exchange bytecode. During serialization of an object all, its data is saved in a file and the objects, which it references recursively in an object graph. When the object migrates to different location and the JVM tries to deserialize it, the deserializer will try to create object instances for that class from the available data recreating the object graph on this machine. In situations where the class does not exist in the JVM, the deserializer will send request to the class loader of the application where it was originally serialized to provide for the class file. Since the class is not available on that machine, the class loader will not be able to return the class.

MobilityRPC overcomes this problem by allowing users to write their own class loaders and configure the deserializer to invoke custom class loader to provide classes needed to deserialize an object. If the classes are not present on the other machine, the custom class loader is configured to retrieve classes from anywhere we program it to. In the case of code mobility, it retrieves classes across the network. MobilityRPC defines a protocol to serialize objects instead of using Java's inbuilt serialization technique. It uses Kryo [11] a more efficient and flexible serializer, and its most important feature being it can be configured to load classes using custom class loader. Compared to other mobility frameworks that use Java RMI for communication, MobilityRPC uses custom defined RPC network protocols for exchanging messages. The most important messages are, Execution Request() - it encapsulates a serialized object wrapped in a Runnable or Callable object [12]. It sends request to MobilityRPC libraries on remote machine to deserialize the object and invoke the run() method in it. ResourceRequest() – this call allows a remote machine to request for a resource (bytecode) from the source machine [12].

ISSN: 0976-3104

IMPLEMENTATION

The execution is carried over two differently configured setups having different testbed setup ups whose performance is measured under the same synthetic workload. A traditional system is one that stimulates a normal distributed processing system, where data across the network is fetched for execution. On the other hand is the proposed system architecture, which uses the mobility RPC framework for offloading the code on the data node and carrying out the execution. The traditional system and the proposed system architecture are given below in the [Fig. 2] and [Fig. 3].

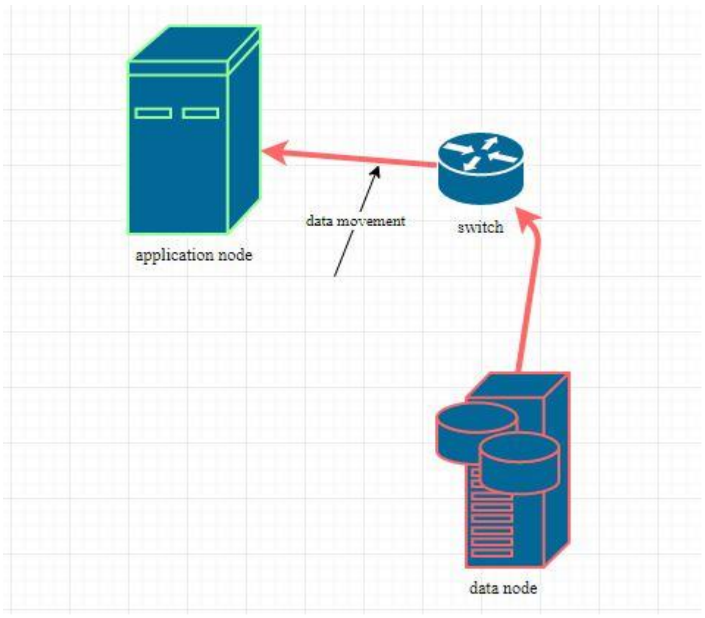

Fig. 2: Testbed1: traditional system architecture.

……………………………………………………………………………………………………..

Testbed1 configuration: Testbed1 is configured in a normal client server configuration. The application runs on the application node and requests for data through the network from the data node. The application is written in Java, it runs through NetBeans, an IDE for running the application, and the database connectivity code is in MYSQL hence uses JDBC drivers to access the database on the storage node. It uses a TCP/IP connection specified by the IP address and port no of the data node to connect and exchange data.

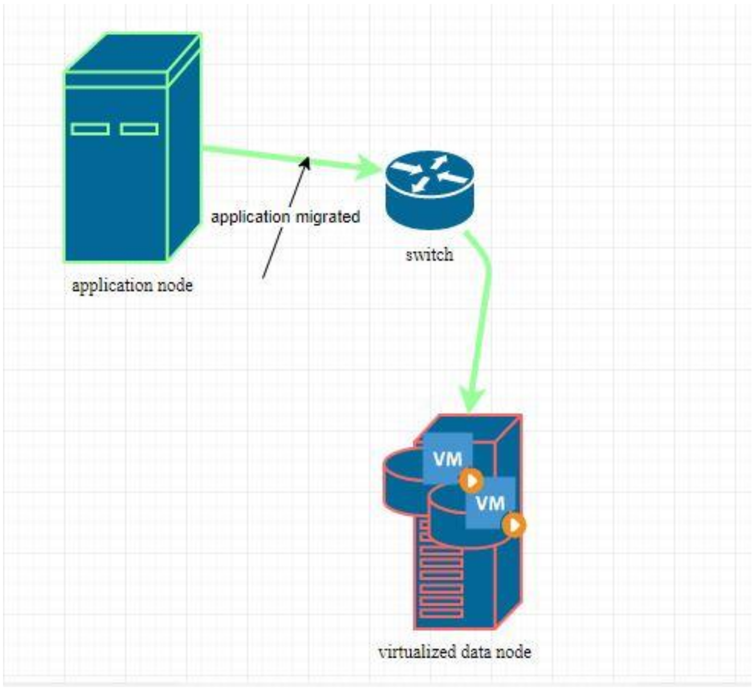

Fig. 3: Testbed2: proposed system architecture.

……………………………………………………………………………………………………..

IIOAB JOURNAL ISSN: 0976-3104

Testbed2 configuration: Testbed2 configuration is similar to that of Testbed1. Here the MobilityRPC framework runs on the application node inside the NetBeans IDE. The MobilityRPC standalone server is running on the data node to receive incoming mobile agents from the application node. The mobile agent on the application node carries the application code to the data node hence MYSQL is able to connect to the database using localhost cause no network is involved and executes on the data node. VMware ESXI 5.5 hypervisor is used to for virtualization on the data node and provide real time performance analysis.

For the stimulation on the traditional system, a sample MySQL employee database is used as the workload. The application is designed to execute query, which requires connecting to the database located on the data node and fetch data. This causes data to be moved across the network affecting the execution time of the application. The query is designed to be data intensive in nature and have high I/O bound requests. For the proposed traditional system, the same application executed except it is embedded on to the MobilityRPC framework. The data intensive part of the application code is migrated to the data node for execution hence stimulating the concept of near data processing or active storage. By moving the application closer to the database, it does have to open remote connections to the database and can directly process it in-situ speeding up the execution time of the application compared to that of the traditional system.

RESULTS

The SQL query executed on the employee database for both the architectures is SELECT * from employees, salaries WHERE emplyoees.emp_no=salaries.emp_no ORDER BY salary DESC, last_name ASC, hire_date DESC. The LIMIT function is used to limit the number no records retrieved. What makes this query data intensive is the ORDER BY clause, which demands the records, returned to be sorted as per the requirements. Since sorting is data intensive and the returned columns are individually sorted resulting in the entire database to be traversed. [Table 1] below gives, us a comparative analysis of the execution time required for the query on traditional system and the proposed system architecture.

Table 1: Execution times for traditional and proposed system architecture

Number of records retrieved	Execution time (traditional architecture)(sec)	Execution time (Proposed Architecture)(sec)
150000	97	98
200000	161	108
250000	205	124
300000	244	144
350000	278	183
400000	345	199
450000	372	235
500000	638	273

From the data in [Table 1], it can be easily concurred that the proposed system architecture is better for executing data intensive tasks and is the way forward. Initially when the number of records to be retrieved are less the traditional system is better because of the overhead associated with MobilityRPC, however as the size of data increases this overhead is compensated. On calculating the average from the above table, a significant 35.36 % reduction in execution time is achieved by implementing the proposed system architecture. The graphical representation of the table is given below in the [Fig. 4]

From the above results, we can conclude that the proposed framework can be integrated in MySQL as an additional feature for executing data intensive tasks such as sorting and traversing through database located on remote machines. It can find application in any database that offers remote database connectivity.

CONCLUSION

After comparing the results of the simulation on the traditional system and the proposed system architecture a 35.36% reduction in execution time is observed which hence proves that the proposed model is more efficient for executing data intensive tasks than the traditional system. Although not applicable in all scenarios, it strengthens the argument of offloading compute logic near the data to decrease the execution time instead of accessing data across the network which increases execution time in case of high I/O bound tasks. The proposed framework can be integrated into data intensive computing for efficiently executing distributed computing similar to Hadoop's HDFS for data processing. Other areas of research that are of interest for this framework would be data mining, where the application is offloaded to the data warehouse and load balancing. Creating a dynamic framework for migrating compute intensive and data intensive tasks on demand and deploying the framework on advance hardware such as SSD's in RAID configuration remains the future scope of the research.

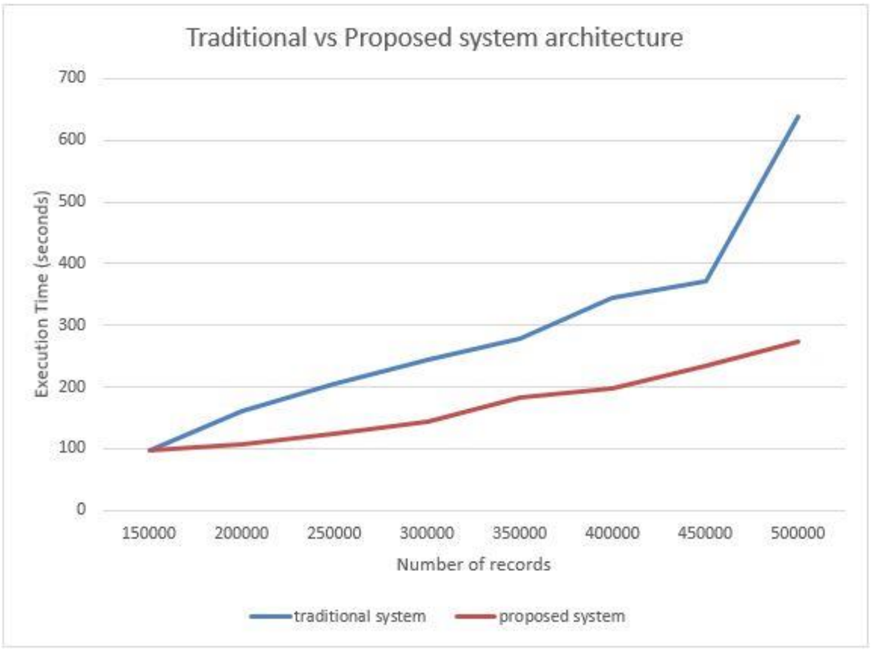

Fig. 4: traditional system v/s proposed system.

……………………………………………………………………………………………………..

CONFLICT OF INTEREST

There is no conflict of interest.

ACKNOWLEDGEMENTS

None

FINANCIAL DISCLOSURE

None

REFERENCES

[1] Acharya A, Uysal M, Saltz J. [1998] Active disks: programming model, algorithms and evaluation, *SIGPLAN Not.*, 33(11): 81–91.

[2] Keeton K, Patterson DA, Hellerstein JM. [1998] A case for intelligent disks (IDISKs), *ACM SIGMOD Rec.* 27(3): 42–52

[3] SV Anastasiadis, RG Wickremesinghe, JS Chase.[2005] Lerna: An active storage framework for flexible data access and management, *Proc. IEEE Int. Symp. High Perform. Distrib Comput*, pp. 176–187.

[4] Ma X, Reddy ALM [2003] MVSS: An Active Storage Architecture, *IEEE Trans. Parallel Distrib. Syst.*, 14(10): 993–1005.

[5] Riedel E, Gibson GA, Faloutsos C. [1998]Active Storage for Large-Scale Data Mining and Multimedia," *Proceedings*, no. February, pp. 62–73.

[6] Joo Y, Ryu J, Park S, Shin H, Shin KG. [2014] Rapid prototyping and evaluation of intelligence functions of active storage devices, *IEEE Trans. Comput.*, 63(9):2356–2368

[7] Runde MT, Stevens WG, Wortman PA, Chandy JA. [2012] An active storage framework for object storage devices, *IEEE Symp. Mass Storage Syst. Technol.*,

[8] Yin S *et al.*[2015] HcDD: The hybrid combination of disk drives in active storage systems, *Proc. - 2015 IEEE 17th Int. Conf. High Perform. Comput. Commun. 2015 IEEE 7th Int. Symp. Cybersp. Saf. Secur. 2015 IEEE 12th Int. Conf. Embed. Softw. Syst. HPCC-CSS-ICESS 2015*, pp. 678–683.

[9] Bare metal hypervisor, [available] https://searchservervirtualization.techtarget.com/definition/bare-metal-hypervisor [20-April-2018].

[10] Mobility-RPC, [Available] https://github.com/npgall/mobility-rpc [20-April-2018].

[11] Mobility-RPC technigies used, [Available] https://github.com/npgall/mobility-rpc/blob/master/documentation/TechnologiesUsed.md [20-April-2018].

[12] Mobility-RPC Protocol Messages [Available] https://github.com/npgall/mobility-rpc/blob/master/documentation/ProtocolMessageStructures.md [20-April-2018].