БАЗИ ДАНИХ, БАЗИ ЗНАНЬ ТА ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 004.773:614.29(045)

Melnyk V.M., Zhygarevich O.K., Melnyk K.V.
Lutsk National Technical University

HIGH PRODUCTION OF JAVA SOCKETS FOR HEALTH CLOUDS IN SCIENCE

Computer clouds are using in health science for its data collections, manipulations and providing security needs in communications to exchange. The clouds distribution data character is using in science applications created to evaluate the data of the health-care. The science programs like medical visualization, genetic and protein conclusions, map-drag therapy and clinical decisions systems of support (CDSS) require high performance messaging libraries with minimum computer and communication spends and the effective utilization of the resources. The high-performance Java sockets (HPJS) encapsulate the needs of message high communications between cloud platforms science applications. HPJS effectively uses the Java socket realization for high-performance inner-process communications. With single-copy protocol, re-usability of the thread and communication overhead reduction, HPJS can use the message exchange in two times quickly to conventional buffered communication libraries.

Комп'ютерні нагромадження даних використовуються в області охорони здоров'я для зберігання даних осіб, їх маніпуляції і забезпечення потреб безпечного обміну. Характер розподілу подібних нагромаджень даних може бути розроблений для застосування в наукових додатках, які розроблені для формування оцінки даних охорони здоров'я. Такі наукові програми як медична візуалізація, генетичні і протеїнові заключення, лікувально-профілактична терапія та клінічні системи підтримки прийняття рішень (CDSS) вимагають бібліотек швидкого обміну повідомленнями з мінімальними комп'ютерними і комунікаційними затратами та ефективним розшаруванням ресурсів. Високопродуктивні Java-сокети (HPJS) інкапсулюють потреби високопродуктивного обміну повідомленнями між науковими додатками для cloud-платформ та ефективно використовують Java-сокетну реалізацію для утворення високоефективного зв'язку між процесами. З єдиною копією протоколу при повторному використанні ниток та зменшенні накладних витрат зв'язку високопродуктивні Java-сокети можуть виконувати обмін повідомленнями в два рази швидше із звичайними буферизованими бібліотеками зв'язку.

Компьютерные накопления данных используются в здравохранении для сохранения данных отдельных личностей, их манипуляции и обеспечения необходимости безопасного обмена. Характер распределения таких накоплений данных может быть разработан для использования в научных приложениях, которые разработаны для формирования оценки данных здравохранения. Такие научные программы як медицинская визуализация, генетические и протеиновые заключения, лечебно-профилактическая терапия та клинические системы поддержки принятия решений (CDSS) требуют библиотек скоростного обмена сообщениями с минимальными компьютерными і коммуникационными расходами та эффективным разграничением ресурсов. Высокопродуктивные Java-сокеты (HPJS) инкапсулируют необходимости высокопродуктивного обмена сообщениями между научными приложениями для cloud-платформ та эффективно используют Java-сокетную реализацию для образования высокоэффективной связи между процессами. З единой копией протокола и повторном использовании ниток та уменьшении накладных расходов связи высокопродуктивные Java-сокеты могут исполнять обмен сообщениями в два раза быстрее с обыкновенными буферизированными библиотеками связи.

Keywords: *cloud platform; high-performance Java sockets; health-care; distribution data; decision systems of support.*

Introduction

Last time the cloud computing has emerged as a computing platform with the main accents of reliability, ubiquity and availability. Computing cloud is defined as a service program support integrated with utility computing conception. Now, the public, private and hybrid models: all are creating to collect a data for different aims, which are equipped additionally with program software for service. They also have platforms and infrastructures for the service performance as utility models [1].

The community of the electron methods involving into health care the utility provided of cloud computing models for bio-medical and health-care data collection, data's ubiquitous availability [2], e-Health Services [3], secure and social health cloud systems [4], where the benefits of cloud's distributed infrastructure are obvious. The cloud computing enriches on the cheaper commodity-hardware running with wide variety of available and distributed resources. New technologies are equipped with commodity-hardware for better multi-task providing through the parallel tasks realization. The newer multi-core commodity microprocessors provide the possibility for electron systems of the health care organizations to use a microprocessor parallelism for the science applications and high performance necessities in health care and biomedicine areas.

Such programs as medical visualization, gen and protein annotation, map drag therapy and CDSS are so good representatives for high-performance computing [4].

Medical visualization applications process human body images (CT-scanning and MRI) and generate the most of data scope. The estimation of this wide-formed data view takes a significant time and needs more than one resource of calculation to achieve the results. In case of high-performed clusters (HPC), image scientific applications divide all data to smaller parts and distribute them over a network to computer. They are naming nodes. These nodes have a parallel work and heavily relay on inter-node and intra-node messaging for a calculated result. HPC's divide and conquer approach essentially reduces the time necessary for health-care and biomedicine diagnostics between related scientific applications.

Usually scientific applications need high-performed clusters to run [5]. These clusters, tied with more funds and low availability, can be out of reach for small research laboratories and individual researchers, but cloud utility-based model can so help here. With cloud-computing enabled with high performance to proceed the information even an individual researcher can run on an ewer scientific applications and perform the modeling at any time from his own computer. Public collecting providers, as Azure Platform from Microsoft [6] and AWS Amazon [7] are already providing their infrastructure for scientific needs. Even so big private cloud platform Open Nebula [8,9] works with open source code can serve the limited scale HPC purpose.

In computation platforms appointed for the research and development, the software has a significant role in the acceptance. More and more scientific applications (like Java .Net for Microsoft) made as program platforms in nearest generation. Java among the popular programming languages, were been adopted with few scientific applications, including medical visualization in heterogeneous environments, spatial and temporal modeling for infection illness and support systems for clinical decisions to make [10-14]. Most of the high-performed message libraries in production [15] are basing on a message-passing interface (MPI). MPI is de-facto HPC-communication standard, compiled in old languages i.e. FORTRAN, C and C++ with close to the minimum support of the cloud computing. Java now is available on cloud platforms like Microsoft Azure [16], Amazon AWS and EC2 [17], Google's AppEngine [18], OpenNebula [8], and still lacks HPC support. Isolated attempt to use Java for HPC made as a result specification in Java MPI 1.2 [19]. However, the deep analysis proves many opportunity areas catered before attempting HPC in platform based in Java. Perspective areas include high-performance inter-node and intra-node messaging middle-ware based on MPI.

Proposed HPJS is one of the libraries to perform high-performance communication between processes by using the implementation of the Java sockets. In the HPJS connection layer the single copy protocol implemented for the extra copy overhead reducing. For asynchronous communication were introduced cached thread pools to provide resource re-usability. HPJS effectively provides the optimization of the computation, better resource utilization, and reduces network overhead for cloud platforms running applications.

Related works review

In HPC area of HPC scientific applications most of the research tends to focus on the core of the application. HPC messaging middle-ware may often be adopting or neglecting as a third-party implementation. Java-based smart-home infrastructure was been proposed in [20] for health-care needs. Several proposed components of HARE engine, require high-performance computation for a prior of the activity recognition to the life-care support services execution and the analysis for long-term activity. HARE presumes that the underlying cloud platform is quit optimized for substantial data optimal processing. Instead of Java socket-based communication, remote method invocation (RMI) or XML-based messaging, HPJS can increase HARE's performance by evaluating of the data activity between a few nodes with the high-performance messaging.

Effectively captured and implemented idea for Java-based high-performance messaging perspective were been realized in [21] as messaging exchange library named MPJ-Express (MPJE). MPJE provides Java's NIO based messaging implementation, but NIO depends from its buffering layer results in computation overhead by involving additional priority byte-copies in its corresponding buffer before sending and after receiving messages. HPJS encounters this overhead to manipulate the bytes directly between scientific applications, HPJS components and sockets following a single copy protocol providing better computation performance and utilization of the memory.

Java fast sockets (JFS) [22] provide the same implementation to HPJS and identified in MPJE catered issues. However, JFS utilizes its own functions though JNI to reduce the copying and implements shared memory protocol for clusters intra-node communication. From other side, HPJS is a Java implementation for commodity hardware HPC over clouds.

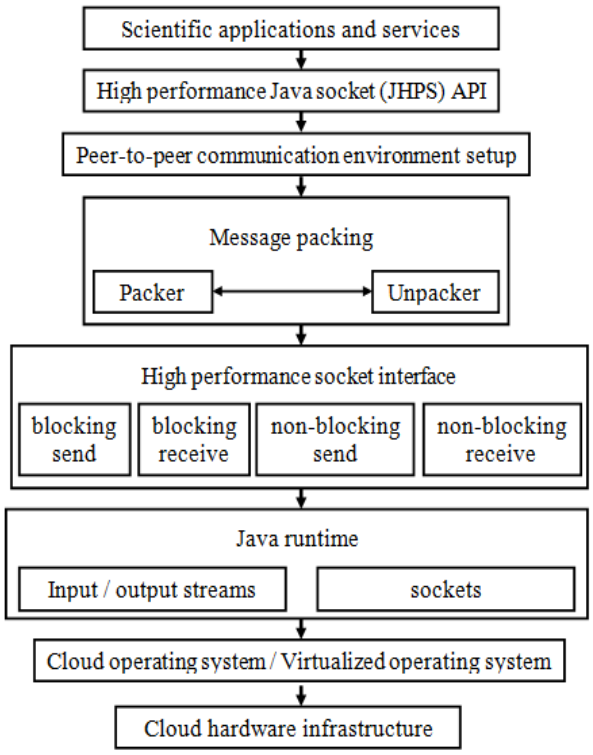

Fig.1. Deployment stack of HPJS

From the looking point of generic high-performance Java-based communication perspective, in [23] Apache's application framework is accepted as a perspective and popular implementation. It uses the Java-NIO implementation for high-performance applications and network high-scalability applications. The MINA framework is not tailor-made for MPI-based implementation but MPI-based implementation can be hold as MINA supports for the synchronous and asynchronous communication.

HPJS architecture

HPJS inherits PaaS-based service model. It has a multiple-level architecture, which can be stretching out as virtualized and non-virtualized environments. Fig. 1 shows the deployment stack with HPJS over the cloud platform. Science programs and services communicate through HPJS API than Peer-to-Peer environment setup executes pre-messaging scripts to initialize the process of HPJS with the all other HPJS information running processes and locations of them. The message pack level packs and unpacks income and outcome messages in the byte form and depending on the communication type. It invokes blocking and unblocking in the communication.

HPJS realization and results

Fig. 2 describes a HPJS execution flow. Every component on the scheme describes its own inputs, outputs and overall interaction of the system. For internal HPJS communication, messages are encapsulating as request objects and response objects for the reusability and better abstraction. For setup the environment, HPJS process is initiating by HPJS-daemon, which loads the configuration file of the machine to provide the initial information about all nodes of the HPS clouds. This information includes cloud-node IP-addresses, port numbers of HPJS-processes and their execution ranks. To use the configuration file all the processes share their unique identifiers (UUID) as resulting parameters for creation of socket objects keyed with a respective identifiers. Every process maintains a socket table containing UUID-identifiers and respective object sockets. This socket table is using for the source identification and the destination objects identification during sending and receiving messages are executing.

Science applications are communicating with HPJS via API, which accepts and returns messages in the byte form. In case of applications for medical imaging, these bytes can represent the partial image or estimation results for the part image. The knowledge of the data sharing is every time encapsulated by the application. HPJS has no knowledge about the data context, takes care for effective inter-process communication. API HPJS de-couples HPJS core realization from the scientific applications. The developer of the new science program needs no knowledge of HPJS internals, all that application requires is conformance to the HPJS API.

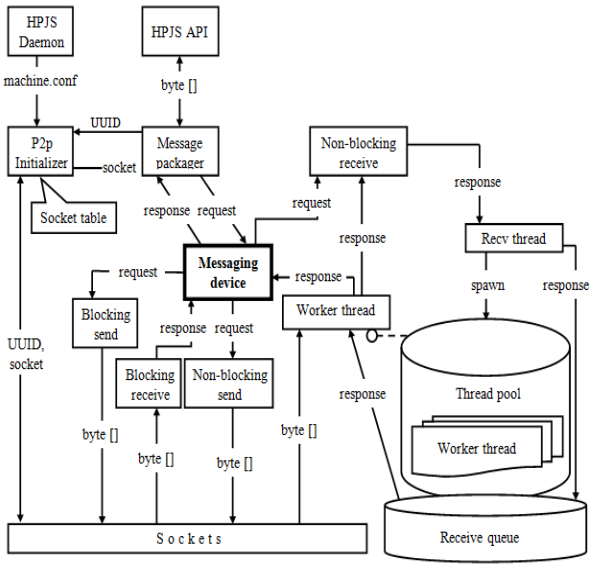

Fig. 2. Execution Flow of HPJS

Instead, to use primitive data types, HPJS performs manipulations over bytes only running single copy protocol opposed n distinct operations of the copy. Single copy protocol copies the array of bytes in one-go ensuring a near native accomplishment. Opposite MPI, specifications to provide the connection between heterogeneous and non-contiguous data custom objects and derived data types are not supporting by HPJS because the sterilization may increase overheads of the performance. From design prospective of HPJS, custom objects can increase the dependency between HPJS core and the science application. The message packer creates objects of the respect and response for components of the HPJS internal interaction. The request and response objects pack bytes for sending or receiving by the socket layer and include the halved actual message: the header and the payload. The payload part is the byte part taken for HPJS from science applications. The header part has a constant size and provides the information regarding the payload part that has the dynamic size (depending of the message).

The blocking of the sending and receiving provides the control of the synchronous communication of the protocols connection, and the operation of the sending is not completing until the messages are all accepted. Non-blocking send and receive facilitate asynchronous protocols for the communication and the non-blocking receive spawns a worker-thread from a cached threads pool to process response without the receive hold in the wait. In the cached thread pool, realization of HPJS re-uses threads and maintains an optimal threads number in the pool in any exception time. This model facilitates running processes to use resources on the cloud node optimally.

With the resource on the clouding computing demand model, HPJS process presumes to have unlimited resources. Instead of the request availability acknowledgements from the process of the receive regarding the payload size, the message has to be sent in one-go. This technique persuades the inter-process network overhead to one send for every message. The blocking receive and non-blocking worker-thread component has to receive the message with one read. However, it evaluates the message in two steps i.e., with the send overhead knowledge or header size, header is read first. The size and type of the payload has to be defining from the header bytes, and afterwards the payload is reading into a byte array.

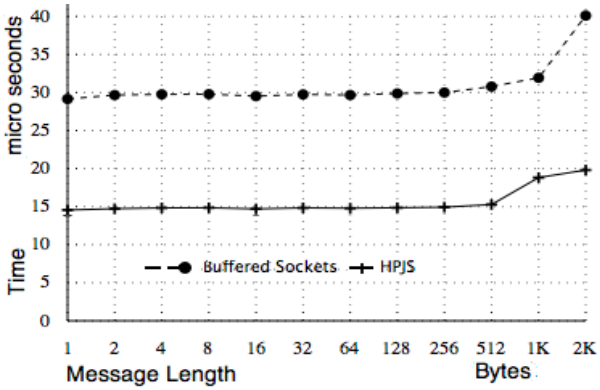

Fig. 3. Analysis of HPJS Performance

Fig. 3 describes the analysis of the preliminary performance of HPJS in contrast with communication device of the buffered socket that uses Java NIO's byte buffer [21] instead of the arrays of bytes. The results show correctly that HPJS performs twice a quickly from the buffered device that provides additional data copies from bytes to the byte buffer. These results were been estimated on a Fast Ethernet based private cloud structure constructed with Core 2 commodity microprocessors and a RAM of 8 GB. The results shown in fig. 3 provide preliminary HPJS proof-of-concept. The right scalability and tests over larger clusters and cloud platforms have to be performed yet.

Conclusions

The lack of Java-based high-performance messaging middleware for scientific applications in health clouds has been the main aim for HPJS. As most of the HPC-based middle-ware utilizes outdated languages and platforms, HPJS presents MPI high-performance inner-process communication built on Java and compatible with most of the known cloud platforms. With its single-copy protocol, cached thread pools and one-go send and receive of the message, HPJS effectively provides the high-performance inner-process communication with optimized computation, reduced network overhead, better performance and resource using. The HPJS evolution includes the better intra-node communication, performance evaluation on clusters with large scale and cloud platforms and integration with different scientific application to complete the true objective of HPJS for scientific health clouds based on Java.

References

1. Armbrust M., Fox A., Griffith R., Joseph A. D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I., Zaharia M. View of Cloud Computing. Communications of the ACM, 53(4), pg. 53-58, 2008.

2. Rolim, C.O., Koch F.L., Westphall C.B., Werner J., Fracalossi A., Salvador G.S. A Cloud

Computing Solution for Patient's Data Collection in Health Care Institutions. eHealth, Telemedicine, and Social Medicine, ETELEMED, 2010.

3. Fan L., Buchanan W., Thummler C., Lo O., Khedim A., Uthmani O., Lawson A., Bell D. DACAR Platform for eHealth Services Cloud. IEEE Cloud Computing (CLOUD), 2011.

4. Wooten R., Klink R., Sinek F., Yan B., Sharma M. Design and Implementation of a Secure Healthcare Social Cloud System. IEEE / ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012.

5. High-Performance Computing Applications. http://www.altera.com/endmarkets/computer-storage/computer/hpc/applications/cmpapplications.html.

6. Microsoft offers HPC on Azure. http://www.itworld.com/virtualization/128231/microsoft-offers-hpc-azure.

7. High Performance Computing (HPC) on AWS. http://aws.amazon.com/hpc-applications.

8. Milojii Dejan Llorente Ignacio M., Montero Ruben S. OpenNebula: A Cloud Management Tool. IEEE Internet Computing, March-April 2011.

9. OpenNebula. http://opennebula.org.

10. TIOBE Programming Community Index for July 2012. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

11. Fedyukin I.V., Reviakin Y.G., Orlov O.I., Doarn C.R., Harnett B.M., Merrell R.C. Experience in the application of Java Technologies in telemedicine. eHealth International. 2002.

12. Drishti: Volume Exploration and Presentation Tool. http://sf.anu.edu.au/Vizlab/drishti.

13. The Spatiotemporal Epidemiological Modeler (STEM) Project. http://www.eclipse.org/stem.

14. Iram Fatima, Muhammad Fahim, Donghai Guan, Young-Koo Lee, Sungyoung Lee. Socially Interactive CDSS for u-Life Care. The 5-th ACM International Conference on Ubiquitous Information Management and Communication (ACM ICUIMC 2011), Seoul, Korea, February 21-23, 2011.

15. Snir Marc, Otto Steve W., Walker David W., Dongarra Jack, Huss-Lederman Steven. MPI: The Complete Reference. 0262691841, MIT Press, Cambridge MA, USA, 1995.

16. Windows Azure SDK for Java. http://www.windowsazure.com/enus/develop/java

17. AWS SDK for Java. http://aws.amazon.com/sdkforjava.

18. App Engine Java Overview. https://developers.google.com/appengine/docs/java/overview.

19. MpiJava 1.2: API Specification. http://www.open-mpi.org/papers/mpijava-spec.

20. Asad Masood Khattak, Phan Tran Ho Truc, Le Xuan Hung, La The Vinh, Viet-Hung Dang, Donghai Guan, Zeeshan Pervez, Manhyung Han, Sungyoung Lee, Young-Koo Lee. Towards Smart Homes Using Low Level Sensory Data. Journal of Sensors, 2011.

21. Baker M., Carpenter B., Shafi A. MPJ Express: Towards Thread Safe Java HPC. IEEE International Conference on Cluster Computing, 2006.

22. Guillermo L. Taboada, Juan Tourio, Ramn Doallo. Java Fast Sockets: Enabling high-speed Java communications on high performance clusters. Computer Communications 2008.

23. Apache Mina Framework. http://mina.apache. org.

Information about authors:

Melnyk Vasyl Mykhaylovych – PhD, Assistant Professor, Assistant Professor of Computer Engineering Department of Lutsk National Technical University. Scientific interests: computing, programming and sockets.

E-mail: melnyk_v_m@yahoo.com

Zhyharevych Oksana Kostyantunivna – Assistant Professor of Computer Engineering Department of Lutsk National Technical University. Scientific interests: computer programming, simulation-based semantics.

E-mail: oz_lutsk@mail.ru

Melnyk Kateryna Victorivna – PhD, Assistant Professor, Assistant Professor of Computer Engineering Department of Lutsk National Technical University. Scientific interests: computational intelligence systems.

E-mail: ekaterinamelnik@gmail.com