ISSN : 0974 - 7435 Volume 10 Issue 15

2014

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(15), 2014 [8768-8774]

The Java virtual machine in a thread migration of distributed application

Guanghui Zhu
Institute of Applied Technology, University of Science and Technology Liaoning, Anshan, 114011, (CHINA)

ABSTRACT

With the rapid development of Internet, the role of the Java virtual machine technology is also becoming increasingly important. First of all, the Java virtual machine for the underlying implementation of cross-platform, for the implementation of the underlying heterogeneous system shielding the details of the infrastructure. Second, the implementation of the Java virtual machine can according to the different application scenarios and the underlying infrastructure do different optimization, and transparent to upper layers at the same time, to achieve the same code compile once, in a different environment to realize the execution efficiency of the optimization. In addition, the Java virtual machine to provide automatic memory management, garbage collection mechanism and characteristics of dynamic link completely, can effectively reduce the complexity of the design and implementation of the large system. The Java virtual machine because of its many excellent properties, are out of pure Java language execution environment, the role of general platform as the next generation. Based on migration of thread are studied in this paper to realize distributed Java virtual machine, on the basis of theoretical issues, the design and implementation of a typical representative system are analyzed, put forward in the Java virtual machine level to realize distributed computing framework based on the migration of threads, realized the distributed prototype single Java virtual machine, and to actually use performance test and evaluation, main work includes: the thread of distributed Java virtual machine migration mechanism; Task migration when the load balancing strategy; Across nodes of transparent access to distributed Java heap; Java thread synchronization mechanism under distributed environment; Under the distributed environment of automatic memory management and garbage collection; Single Java performance test and evaluation.

KEYWORDS

Distributed computing; Parallel computing; The Java virtual machine.

INTRODUCTION

With the continuous development of the Internet, the importance of distributed computing is becoming more and more prominent. The Java virtual machine for the upper application blocks at the bottom of the computing platform implementation details, very conducive to the development of distributed system based on heterogeneous infrastructure. One Java multi-thread programming model and integrating object-oriented model, provides a complete set of effective mechanism of multithreaded programming. Due to native multithreaded Java program, as well as thread between low said features, this paper argues that can is the smallest unit of execution context with Java thread, to break up a Java program. Due to the Java virtual machine for the entire life cycle fully managed, Java thread based on Java virtual machine, can be a good Java thread transparent migration, the Java program into a distributed parallel programs, can effectively improve the parallelism of the program and performance. Java memory model and the automatic memory management mechanism to simplify memory management of distributed system is helpful.

Based on the in-depth study of distributed Java virtual machine, on the basis of theoretical issues, designed a distributed prototype singleJava Java virtual machine, to the upper application provides transparent distributed infrastructure abstraction layer. On the basis of singleJava, Java programmers can completely transparent like Java virtual machine in the single node write debug program, compiled Class files directly after the completion of release to singleJava execution. SingleJava to implement the task based on the migration of threads concurrent tasks migration and parallel execution of distributed computing systems.

SingleJava system source code to use and modify the open source Java virtual machine implementation JamVM Class file parsing function, single node execution engine. In use, modify the JamVM basis function on the basis of the above, realize the distributed Java virtual machine prototype based on thread migration. As shown below is singleJava compared with single node architecture of the Java virtual machine. Single node the class loader is responsible for the Java virtual machine access to Java class bytecode. Get Java classes, analytical, links, and initialization, Java classes are divided into two parts, one part is the static attribute value preservation area, the other part is the definition of properties, methods, area, the static attribute values stored in the Java heap, all attributes and methods defined in the MethodArea. Invoke the execution engine reading Method, after the completion of all methods defined in the Area began to implement a Java Method, will be calculated during the period of the intermediate results stored in thread stack, instantiation of the object in the Java heap. As shown in Figure 1.

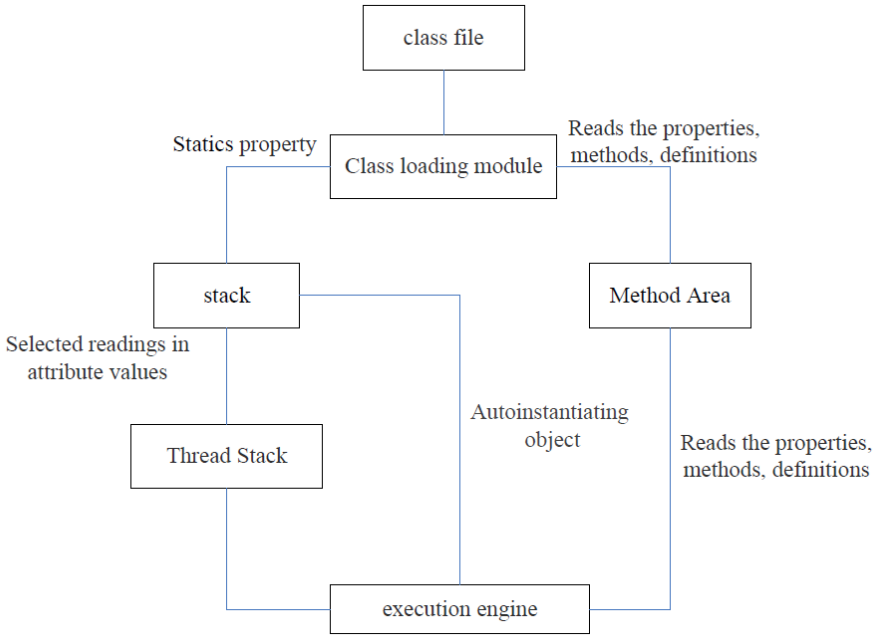

Figure 1: Java virtual machine (single node system architecture)

SingleJava split the Java virtual machine execution engine and the Java heap, the execution engine distributed to multiple nodes, can execute multiple threads at the same time. Local heap on the various nodes communicate with each other, form the global distributed Java heap, transparent execution engine provides transparent access across nodes of the pile.

DESIGN CONCEPTS AND SYSTEM ARCHITECTURE

Design concept

To provide completely transparent to the upper distributed infrastructure, singleJava design based on Java distributed general computing platform, on the one hand, let Java programmers from writing multithreaded applications, launch applications, to finally start the application execution can be completely like Java virtual machine on a single node,

completely managed by singleJava thread distributed execution. SingleJava, on the other hand, in an attempt to support zero modifying legacy code executed directly, do write once compiled, direct execution. To this end, the singleJava adopted code dynamic deployment mechanism and management share data by using distributed Java heap.

Using distributed Java heap can let all nodes are like access to the local Java heap access global objects in a distributed environment, global data synchronization when reading and writing done, ensure the integrity and consistent with the global object. SingleJava belongs to tight said in distributed computing systems, system communication mode between internal nodes, according to the analysis of the communication between nodes more frequently, improve the request processing throughput between nodes, has a key effect on the system overall performance. Therefore, singleJava USES the request processing thread pool technology, singleJava management mechanism design is concise, using a single MASTER node architecture[2].

System architecture

SingleJava system architecture as shown in Figure 2, among them, the single MASTER nodes are responsible for the management of the cluster, the WORKER node contains execution engine, is responsible for the implementation of specific Java code. All maintenance WORKER on the MASTER node information, system initialization when the WORKER nodes to the MASTER node initiated the registration request, record and maintain all the WORKER MASTER node information, and the MASTER node is responsible for the management of WORKER node.

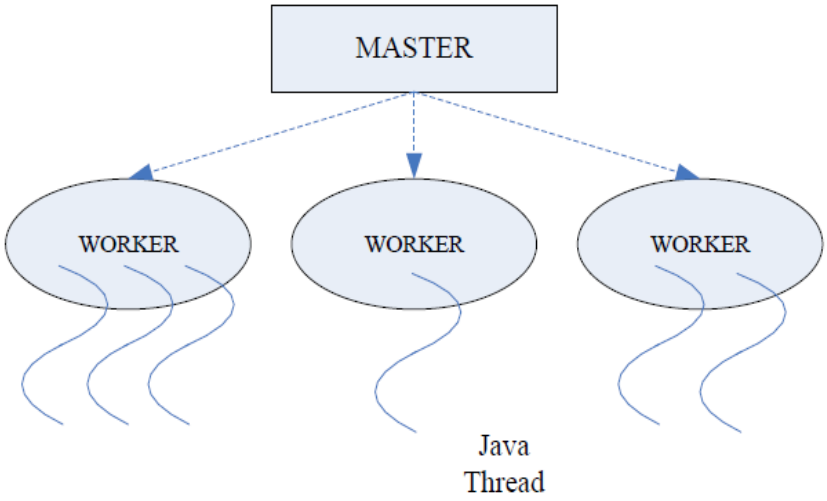

Figure 2: SingleJava system architecture

After completion of system initialization, triggered by the user in the MASTER node from the command line Java application execution MASTER will first read the main class of the class files, to obtain the main class class file sequence of bytes, after the completion of the reading users with command line parameters, the generated command line parameter sequence of bytes, after give priority to the Reference class distribution. Last of all the WORKER nodes choose the earliest registered nodes, launched the REQ - the MAIN - a TASK Request, the Request contains the MAIN class file sequence of bytes, command line parameters, and the MAIN class Reference. After receipt of the Request the WORKER nodes will parse the main class sequence of bytes, generate ClassBlock structure. Launched an REQ_NEW - THREADRequest the MASTER node, after the completion of the MASTER node, after receipt of the Request for the Thread to create a new Thread description structure, and distribution of global Thread identifier Threadid, finally will Threadld SUCC responses as RSP reply to WORKER nodes. After WORKER node receives RSP - SUCC response, find the main class ClassBlock structure.

IMPLEMENTATION

SingleJava adopts the modular system design and code organization mode to break down the complexity of the system. Top-down divides the system into several modules, step by step to segmentation system, organization and packaged [3]. SingleJava in accordance with the rules of good design, said the intensive content of the code to a module, and divide the development tasks. Interact through fixed call interface between modules, in addition to the internal implementation of the module's internal design for free play.

SingleJava code in accordance with the six modules, three levels to organize, as shown in Figure 3.

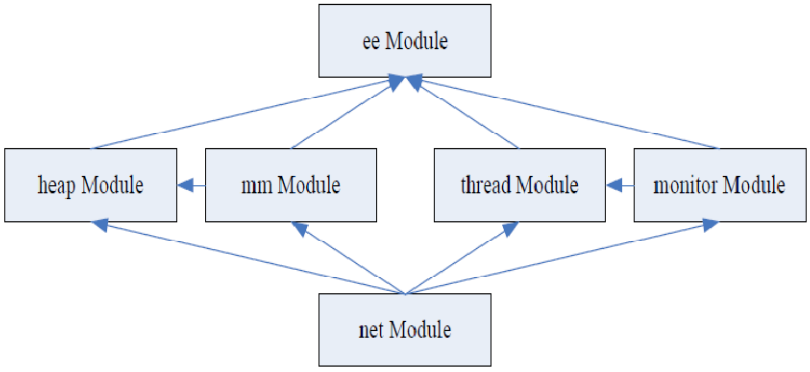

Figure 3: SingleJava module organization

Distributed thread migration mechanism of Java virtual machine

Thread creation and migration work are done by threading module, the threading module principle of Java virtual machine. The start () method, the Java virtual machine started to create a new Thread, first create a Thread to describe structure, link to system Threads linked list, and then create a new thread, Thread and private variable thread_self pointing at the newly created Thread structure. All completed, call execution engine start executing Thread.

SingleJava threading module in a single thread management and the mechanism of the same node, on this basis, the realization of parallel computing based on thread migration: Java program execution ThxeacLstartO method need to apply to the MASTER node to create a new thread, the WORKER nodes selected by the MASTER node, moving a new thread to execute the WORKER nodes. Threading module need to realize the function, on the basis of the implementation of tasks at the same time of the load balancing strategy[4].

The task migration of load balancing strategy

System to the WORKER node number of the current thread of execution at the same time, the basic measure for calculating load when choosing the WORKER to task migration, first of all, according to the WORKER the number of nodes in the current thread in ascending order, the smaller the number of threads of the WORKER nodes the higher priority. Based on this, will give priority to the recently completed a thread of the WORKER nodes distribute new tasks. The WORKER nodes REQ_THREAD - a TASK request processing functions can be carried up a new thread, when after the completion of the new thread execution, request to report to the MASTER node through REQ_THREAD - FINISHED thread the end of the message, and indicate the complete thread Threadld in the request. MASTER node according to Threadld delete records in the list of Threads, thread will WORKERS number of Threads in the list of minus one, the latest completion time and update the corresponding[5].

Transparent access to distributed across nodes Java heap

Namely heap object module module, responsible for the execution engine provides transparent across the node object access, regardless of the object is in the local or remote on physical nodes on physical nodes, object modules provide a unified access interface to the execution engine. JamVM, for example, the principle of single node object module of the Java virtual machine as shown in Figure 4, is in the charge of this Class files, and save in the form of a byte array. Analytical, links, and then in turn calls the initialization function, to generate the ClassBlock structure (the basic information of the save the class definition, attribute definition, method and constant pool) and the Statics structure (used to hold all the static attribute values), and provide the ClassBlock pointer to execution engine, used to access the ClassBlock structure[6].

SingleJava Object module based on this, to achieve the transparent across nodes access the Class/Object. Includes the following contents: across nodes class loading, the WORKER nodes from the local system class for class files, to the MASTER node requests for user classes class files; Using a global identifier instead of the pointer as the Class/Object global references; Using a global reference implementation of global Class/Object to read and Write, use the Write - Invalidate agreements to ensure data consistency and integrity[7].

Java language native support multithreading, Java virtual machine memory model is the design in the perspective of thread. The data in the Java heap is accessible to all threads, Java heap was also called main memory, each thread has its own private memory, known as the working memory. Each method is called began to start execution will create a method, is used to store method of the call stack, call parameters and local variables and the operand stack, pressure into the thread of the call stack when the method returns, the method frame has been playing the stack.

SingeJava using Java virtual machine memory model the characteristics of low coupling between threads, thread work memory are independent of each other, do to the Java heap distributed implementation. All Object/Statics access need first access to the local heap, if local heap mismatch, is by visiting the Object/Statics OWNER node local heap, access to data, all nodes in the respective local heap constitutes the distributed Java heap[8], and its structure as shown in Figure 5.

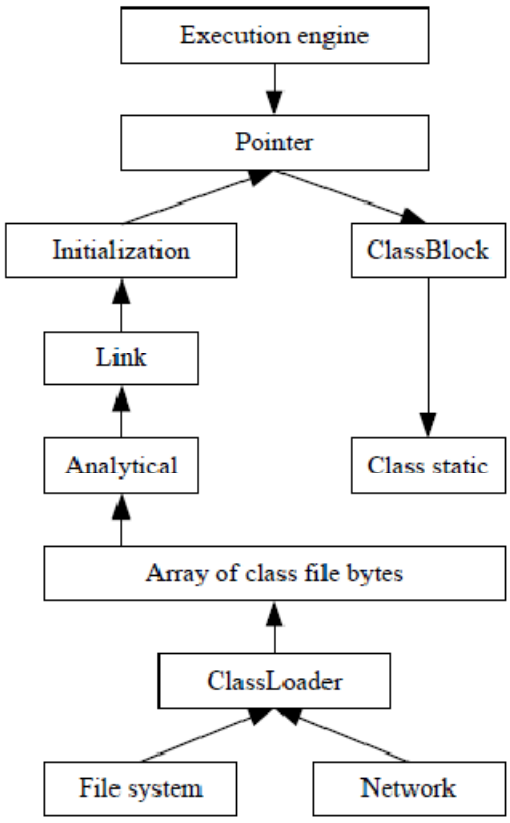

Figure 4: The main principle of JamVm object module

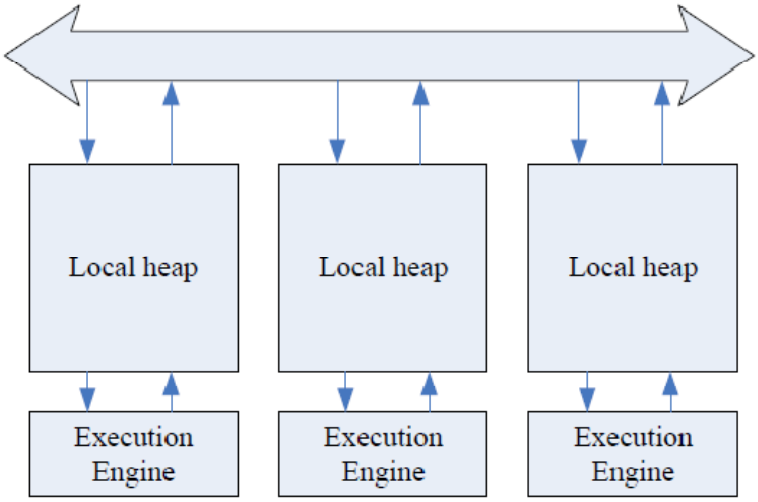

Figure 5: Java virtual machine memory model

An Object/Statics OWNER node with the following rules to determine: if the Object/Statics has not taken any write operations, then the Object/create node is the OWNER of the Statics; If the Object/Statics have been executed write operations, is the latest to perform the write operation is the OWNER of the node.

Distributed Java heap through the Write - Invalidate agreements to protect the data consistency and integrity.

Java thread synchronization mechanism under distributed environment

Java language native support multithreading, and have perfect thread synchronization mechanism, including: Suspend - Resume mechanism, Synchronized mechanism and Wait - Notify mechanism. SingleJava achieved in a distributed environment, across nodes of a full set of thread synchronization mechanism. Java language native support for multithreaded programming, provides a complete thread synchronization between the mechanism of strong, Java thread from creation to

end with a total of six kinds of state, are: NEW, RUNNABLE, BLOCKED, WAITING > TIMED - WAITING and TERMINATED. The meaning of their status as shown. Threads in the above the conversion relationship between each state as shown in Figure 6.

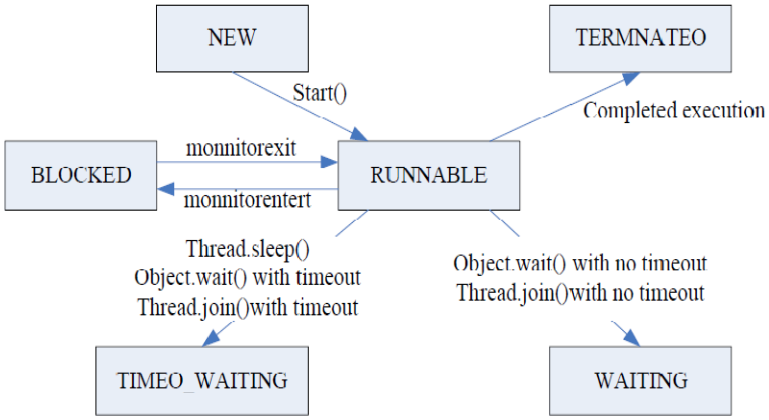

Figure 6: Java thread state transformation

Under the distributed environment of automatic memory management and garbage collection

The memory management module is responsible for the distributed Java heap space allocation and garbage collection. JamVm virtual machine to achieve the first matching algorithm based on the heap space management and recycling mechanism based on the reference count. SingleJava using original heap management algorithm to manage node local heap space, on this basis to realize the distributed garbage collection. Distributed garbage collection in accordance with the local garbage collection and global scope is divided into different garbage collection two levels[9].

The WORKER nodes at startup that is according to the system configuration, to the system requests a piece of memory space, and save the heapend heapbase starting position and end position. Object/Statics are stored in a pile of local, local heap memory allocation interfaces provided for other module, local in the heap memory blocks allocated in accordance with the 8 byte alignment. Local heap by a mutex heaplock is responsible for the protection, each time allocated from the heap space to obtain the lock, to start operation, after operation to release the lock. As shown in Figure 7.

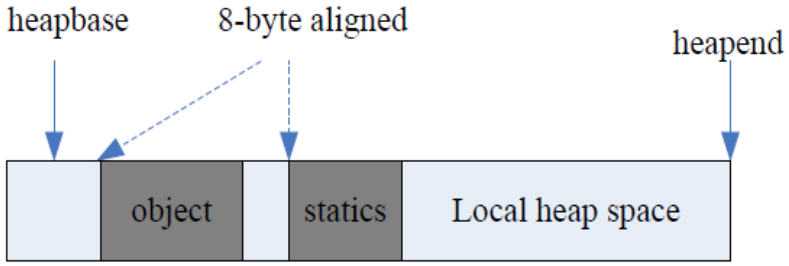

Figure 7: Local heap space

Java virtual machine to provide the upper rubbish, is responsible for automatic recycling useless heap space. There are two kinds of recycling the timing of the program is running, the first is triggered by the user, the user by calling System. GcO approach requires the Java virtual machine began to start recycling. Another kind is triggered by heap space allocation failure, namely the user attempts to new space allocated on the heap space to store the Object/Statics heap space is insufficient, will trigger the system for garbage collection.

Single Java performance test and evaluation

Performance testing using Virtual Box Version 4.2.4 r81684 Virtual machine, a total construction of the eight node configuration in the same experiment. Multi - threaded Benchmarks will be divided into two parts. The first part is the core algorithm is commonly used in large-scale application of the test. Test items were Series, LUFact, SOR;The second part is the simulation test of large-scale applications, in order to improve the compatibility of test case, test omitted the I/O and graphical interface, highlighting the Java execution environment performance test[10].

The above performance test data show that singleJava system computationally intensive parallel tasks in the implementation of the performance is better.

CONCLUSION

Based on the distributed system are summarized and on the basis of current situation of the development of the Java virtual machine, puts forward the layer in the Java virtual machine based on thread migration implementation of distributed computing framework, and further analyzed the distributed application requirements and technical advantages of Java virtual machine. On this basis, this paper implements the distributed singleJava Java virtual machine prototype based on thread migration. By singleJava system architecture and the analysis of test data, this paper reflects on the singleJava in the design and implementation of the existing shortcomings. In the future work should focus on how to design the WORKER nodes of main preparation mechanism, let the failure of the WORKER nodes automatically withdraw from the system topology and choose to substitute the WORKER nodes continue to execute computing tasks, and on this basis to realize the system topology structure of dynamic management, improve the reliability of the whole system and fault recovery capability.

REFERENCES

[1] Von E Thorsten; Active messages: A mechanism for integrated communication and computation, Conference Proceedings - Annual Symposium on Computer Architecture, 256-266 **(1992)**.

[2] A Baratloo; Charlotte: metacomputing on the web, Future Generation Computer Systems, 559-570 **(1999)**.

[3] M Baker, B Carpenter, G Fox, S H Ko, S Lim; mpiJava: An object-oriented java 98 interface to MPL Lecture Notes in Computer Science, 1586: 748-762 **(1999)**.

[4] M Baker, Southsea; mpiJava: A Java MPI Interface. http://acet.rdg.ac.uk/~mab/Papers/EuroPar-98/, 1998, Accessed on June **(2012)**.

[5] EPCC, The Java Grande Forum Benchmark Suite, http://www2.epcc.ed.ac.uk/computing/research_activities/java_grande/index_L.html, Accessed on June **(2012)**.

[6] P Michael, Z Matthias; JavaParty - Transparent Remote Objects in Java, Concurrency Practice and Experience, **9(11)**, 1225-1242 **(1997)**.

[7] S Deering; Host Extensions for IP Multicasting. IETF RFC 1112,1989 [27] J Archibald, Jean-Loup B.Cache coherence protocols:evaluation using a multiprocessor simulation model, ACM Transactions on Computer Systems, **4(4)**, 273-298 **(1986)**.

[8] P Jouppi; Cache write policies and performance, ACM SIGARCH Computer Architecture News, **21(2)**, 191-201 **(1993)**.

[9] N Sankaran; A bibliography on garbage collection and related topics, ACM SIGPLAN Notices, **29(9)**, 149-158 **(1994)**.

[10] Hans-J, Boehm; The space cost of lazy reference counting, Proceedings of the ACM SIGPLAN-SIGACT Symposium, 210-219 **(2004)**.