Dedan Kimathi University of Technology

Security in Distributed Operating Systems

Brian Manoti Okinyi

C026-01-1210/2016

Operating Systems II – ICS 2208

Mr. Morris Kaburu

November 27, 2017

MEMORY MANAGEMENT

The distribution of all data in the cluster as far as possible is one of the challenges faced in designing Distributed OS. Therefore, simple object structures are not the only thing shared by Distributed OS but also code that is usable by all nodes. Because of the difference in system critical components and hardware characteristics, they have resulted in cluster that cannot be shared if all the data are to be put in the same cluster. At the start for every node, facilities for the boot process or for network communication must be exclusively available if it exists, before it is able to join the cluster. In providing the best runtime performance, system or any hardware specific components such as network adapter which is bounded statistically are implemented in the local kernel of Distributed OS.

Following the startup process, all necessary drivers and small subset of system tools are initialized by local kernel before the node can join the cluster. Because of drivers and memory management of the local kernel are exclusive for every node and not shared in the cluster, every node and its local kernel are comparable with a single station system. On the contrary, the distributed kernel is completely distributed compared to the local ones and thus, shared by all nodes. The distributed kernel will be loaded from the shared memory of the cluster right after the boot process and initializing of the local kernel. Cluster can be loaded from a special server (so-called boot server) or a separate storage medium such as CD-ROM, if there is none.

The implication of one shared memory management for all nodes in the cluster is done by the distribution of the distributed kernel. Therefore, the view of the shared memory that is the single system image, and the usage of the same code and data structures of the distributed kernel are being shared by every node. An overview of the memory management of local and distributed kernel will be given in the next two paragraphs. Furthermore, the distributed kernel and its data are under control of Transactional Consistency to make sure of the guaranty of correct functionality of the cluster, especially of the distributed memory management.

**Startup process of Distributed OS**

**Local Memory Management**

For Distributed OS, the local memory management place the maintenance of physical pages of each node, page tables and also offers system or driver programmers memory allocation of two memory pools. These pools are placed in between the distributed kernel and the local one (refer paragraph II.C). Java is the well-known new interface dedicated to programmers to allocate memory. In contrast to Sun Java, another internal structure of its objects is used in Distributed OS. Primitive data types (scalars) and references are objects that are separated in Distributed OS. References to objects that separate these two regions are stored below and scalars above.

The opportunity to store scalars at another location in memory is a specific feature of Distributed OS outside of the proper object. Best performance and a fast startup process are offered through these so-called indirect scalars that are not used in the local memory management which keeps it small and avoiding unnecessary complexity. Information on the structure and usage of indirect scalars are shown in paragraph II.C. Creation and the administration of page tables for the logical address space are other functions of Distributed OS. It is inappropriate and impossible to create and store all page tables for the entire 64-Bit address space with the current capacity of memory cluster nodes using a 64-Bit architecture. Thus, the local memory management supply an on-demand allocation and creation of new page tables which keeps the set of page tables smaller at startup. Other than that, if the matching address space is unused, parts of page tables can be removed to cleanup memory. The memory management reserves at startup a specific logical address space to map and access all page tables, due to the allocation and creation on demand that has been mentioned before.

**Local memory management**

**Object structure**

Not only that, page tables can be differed and cannot completely be shared if each core could use other section of section memory with regards to the mechanism of Transactional Consistency. Further research will focus on the complete support of multicore cpus mentioned before.

**Distributed Memory Management**

Distributed OS possess a local and a distributed memory management as mentioned previously and the latter is implemented in the distributed kernel and completely distributed in the cluster. An additional communication facility between both kernels is necessary since two kernels exist in the system which are the local and the distributed and usually, it is fact that not every driver or system component can be shared by all nodes. That is why Integer-Interface is used by Distributed OS with own data buffers to communicate between the local and distributed kernel widening the scope in which code can be shared by nodes of the cluster. To illustrate, the distributed kernel of Distributed OS can implement drivers too. Also, several consistency models can be supported by Distributed OS for its shared data and for this purpose; memory regions are the result of logical address space division. Each region is under control of one consistency model which in size of 512GB thus; the upper 25 address bits specify the consistency which is the same with the upper hierarchical level of page tables. (Hardware details are shown in chapter IV). A fast classification of the used consistency at runtime is offered by this design. Transactional Consistency is used by Distributed OS as default consistency which covers the distributed kernel, the shared system functions and all user application and their data.

**Distributed memory management**

The allocation of memory and object creation in their corresponding memory region is managed by the so-called allocators. They are objects themselves spanning certain address region inside of a memory region and are subdivided into two parts (see figure 5). One part has new objects with their respective references as well as direct scalars and the other has all indirect scalars. As a result, the capacity of the selected allocator is reduced by each memory allocation until its capacity is exhausted or the allocator object is cleared in memory.

To guarantee the safety of the type system, all object references are normally under the control of Transactional Consistency thus, every allocator has to be always created in a memory region and being under its control. Secondly, the needed consistency model is able to use by the programmers which Distributed OS features the opportunity for programmers to use weaker consistency model for noncritical data with no cause danger to the system. How we can loosen up this requirement without putting at risk for the runtime system is our future concern. Programmers of application or system tools can make their own allocator avoiding latent collisions amid different applications and their objects for exclusive memory usage (e.g. false sharing).

**Allocator object structure**

Transactional Consistency

In Distributed OS, all computations are encapsulated in transactions if to be compared with the well-recognized processes of other operating system. Transaction results are written to memory and the original state of a page is preserved by a shadow copy. All shadow copies are restored and the transaction will be restarted automatically if in case of an abort. The addresses of their written pages are sent to all nodes of the cluster at the end of a transaction. Therefore, the nodes can detect collisions between competing transactions. A concurrent transaction has to be restarted if it has read one or more modified pages. The local memory management will note all of the set pages which are read or written which requires marginal software action. No locks are needed to access memory or shared objects for Distributed OS during executions because it uses an optimistic synchronization, allowing transactions to proceed locally and its validation at the end of it.

III. MULTICONSISTENCY

Several consistency models are supported by Distributed OS which kernel or system programmers are the implementers and each consistency covers a separate logical address space which has a size of 512GB. Distributed OS has per default three consistency models: a local, transactional and a special one for memory mapped IO which can be used by device drivers to map their device memory. The preset for all references in Distributed OS is Transactional Consistency where it inhibits invalid manipulations of the kernel runtime structures of Distributed OS or the type system of it. Additionally, weaker consistency models for references will be examined by us without putting functions at risk. Distributed OS is alert of the access of every memory and delegates it to the respective consistency protocol where it can function according to its requirements. This will make possible of the implementation of new consistency models in Distributed OS without complex adaptations in system functions.

IV.PERFORMANCE ISSUES

One of the main functions of every memory management is memory allocation in the perspective of Distributed OS, memory allocation and object creation are included too. Therefore, the time to allocate memory and the creation of new objects have been measured by us including all procedures for a new () in Java. The same procedures implemented in Sun Java on the same hardware have been compared with the results that we have. A selection of the results that we have gathered listing the time to create a simple object is showed in the following overview which contains no additional fields or references:

**Measurements of memory allocation and object creation**

In Sun Java, The Measurement of a new () includes automatic garbage collection and etc. as additional functionality inside the Java Virtual Machine. On the other hand, further functionality with regard to distribution and consistency is included in our object creation and memory allocation. In addition, the potential for our memory management in Distributed OS is some optimizations where it will reduce the average time of a new().

V. GARBAGE COLLECTION ISSUES

An efficient implementation of a garbage collection and working parallel over all nodes in the cluster are two of our next challenges thus, every creation of objects with its reference in the according object descriptor have to be noted by the memory management which is part of the system information. Therefore, a reference will be contained by all descriptors to a special container with a special consistency that is, the so-called relaxed backpacks which references of all instances of a certain object are included. Objects are traceable in memory as the basis of this implementation. Garbage collections could be implemented in several ways and one of the ways is actually uses the type of information of Distributed OS where relations and dependencies for an inspected object are to be identified. Hence, the possibilities that we have to analyze all kinds of references to any object. It is determinable whether an object is garbage or not with kind of information and another possibility to identify garbage is simple by using an offline garbage collection which a special cluster-PC (so-called page server [3]) is implemented in it. Owing to the possible collisions during the access to the relaxed backpacks, it will be stored in a specific consistency model conforming to a reduced semantic that guarantees each node the same content but in possibly different order.

VI. PERSPECTIVES

To better an efficiently working garbage collection for shared memory, several consistency models for Distributed OS will be designed and implemented, supporting system and application programmers and their implementations to achieve best runtime performance. The strongest consistency model in Distributed OS is Transactional Consistency which guarantees the type safety thus no one can break it (in malicious intention or unintended) and cause danger to the runtime system. Weaker consistency models are available for this reason for primitive data. The impact on the performance when using weaker consistency model will further be examined for indirect scalars of objects. Moreover, to relax the consistency restrictions for references can be advantageous for some cases. When references stored in a consistency area with a weaker consistency as Transactional Consistency, it could be useful. On the other hand, Distributed OS type system has to be protected and thus, the focus is on how we could implement weaker consistency for references and in which cases it is useful and uncritical.