Software Architecture for Scalable Computing Systems with Automatic Granularity Selection of Executable Code

Mikhail Pavlov
NPP SATEK plus LTD
Rybinsk, Russia
pavlovma007@gmail.com

Alexander Petrov
Soloviev Rybinsk State Aviation Technical University
Rybinsk, Russia
gmdidro@gmail.com

***Abstract*—The problem of developing software architecture and its platform implementation for scalable cloud services is addressed in the paper. New scheme of distributed software developing and executing is presented with argumentation and main principles behind solution. Performance evaluation of one of the platform components (data storage) is described.**

I. Introduction and related works

Modern software systems could be described as heterogeneous, distributed, high load services. The growth of client devices number and pervasion of information technologies in our life leads to the problem, that extensive computing power capacity couldn't be a foundation for future processing systems. New software platform and architecture for such application as IoT and big data with data mining is urgent. It is required, that such software tool takes into account demand of modern network architectures and cross-platform software with existence of legacy code. In general we need new approaches to application design.

We present new cloud virtual machine (VM) with multiple programming languages support and automatic granularity selection of executable code. We describe main principles behind our VM architecture and demonstrate some preliminary experimental results in part of data access solution. Like some software platform development project, our project involves questions of the structure of executable code, its execution, optimization, storage and data access. Our virtual machine could be classified as PaaS in terms of category of cloud computing services.

Some of a previous research in new approaches to high load cloud services and high performance software application design, that comparable with our work, includes: [1] describes parallel programming models and run-time system support for interactive multimedia applications, author proposes typical software patterns for parallel software application design. In [2] the main attention is paid to implementation of parallel VM based on functional operation basis. [3] which propose method of automatic extraction of pipeline parallelism for embedded heterogeneous multi-core platforms. [4] poses a solution for cloud data processing service design problem and propose new programming language, compiler, runtime and new data representation and storage system.

Previous research in concurrent data access methods and thread safe data structures, that have some common parts with our work, includes: [5], which considers the problem of runtime parallelization of legacy code on a transactional memory system. In [6] authors address a lock-free algorithm for concurrent bags. The [7] article address a question of could software transactional memory make concurrent programs simple and safe. [8] describes a hierarchical transaction concept for runtime adaptation in real-time, networked systems.

To sum up above said, there exists numerous related works which are devoted to different aspects of distributed, high load services development. But in contrast with our work part of them address only some problems of such system construction, part of them lack of conceptual framework and architecture model, and others don't address a problem of automatic code parallelization.

We compare our work with some top of the art software tools in detail in the next section of article.

II. Main principles behind proposed architecture

A. Supercompilation by evaluation during normal system operation

We base on such use case scenario:

1) Developer create a solution for his task in sequential program (single thread).
2) Virtual machine automatically split the code of program in fragments of optimal granularity for given computation nodes.

ISSN 2305-7254

3) During its operation system collect information about different types dependencies and optimize parallel structure of program.

One could compare this idea with supercompilation by evaluation technique [9]. But from such point of view our approach is different in that VM do optimization all the time, during normal system operation, not in the special stage of system development.

B. Program code fragmentation

We represent software application as a process of data and operation graph construction.

By code fragmentation we mean process of program's decomposition on two parts:

1) Program structure part (PS) that describes dependences between data and operation. Not all dependences could be extracted from source code of program, so we need special runtime for dynamic dependences extraction.

2) Program model part (PM) – operation execution approach and it's properties such as:

- which code fragment should execute asynchronous;
- does this operation lazy, strict or background;
- where this operation should be deployed;
- should me decompose this operation on suboperation or not.

Answers on these question depends on several quality criteria, such as:

1) total execution time minimization;
2) minimal computation resources occupation;
3) minimal price for cloud's resources using;

and so on.

C. Intensive transactions use

Virtual machine should solve a task of computation balancing and data distribution. From our point of view data distribution task require an existence of transaction manager in system. Our approach suggests intensive using of transaction and even more using a hierarchical transaction storage. Transactions help not only to save data consistence, but also increase system performance.

D. Practical approach to system exploitation

It is quite common that software developers hard-code program structure (and so a scheme for domain problem solution searching) and program model (and so limit possible computation methods). After that changes in program structure and model are expensive.

In current practice control over software system passes after the creation into the hands of deployment and support departments. At that moment feature and bug request begin to appear, but nobody except programmers could respond to these request. Either way during the operation period program structure and model are developing. We propose, that a practical approach to system exploitation is to pass some part of control over software to VM, that could collect information to adapt program structure and optimize program model in terms of code granularity and lazyness and so on. In such approach one could imagine, that after long period of software application operation VM collect enough information about PS and PM to generate new implementation that will have higher quality in terms of selected criteria.

Main principles behind proposed approach are shown on Fig.1.

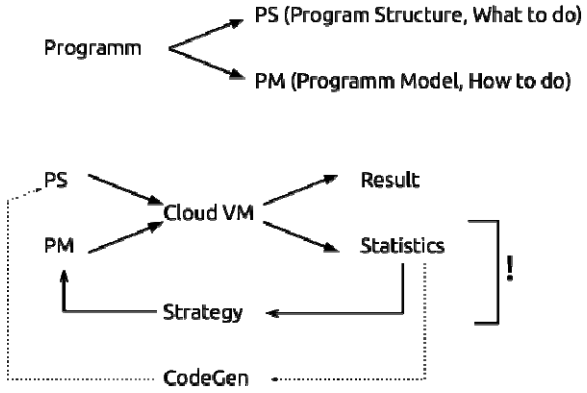

Fig. 1. Main principles behind proposed architecture

III. SOFTWARE ARCHITECTURE FOR SCALABLE COMPUTING SYSTEMS

To realize described approach we propose new architecture that based on several requirements and assumptions:

1) All data are in special storage – TVM (transaction value manager). TVM is hierarchical transaction storage. In fact TVM is only interface that provide access to data within hierarchy of transactions and it could be implemented upon different data storage platforms.

2) Program split into fragments of different granularity. Fragment is a lightweight thread. Fragment could call thee system to execute other fragments. Computation is a fragment that is executed within transaction.

3) Fragment could be external and internal against the system. External fragments are the code that executed in separate processes. Internal fragments are executed in the system's processes. External fragments export additional function to system (like a plugin system).

4) Fragments could work only with data from TVM and all version of processed data and new data are saved in TVM.

5) The system decides when and on which computation node start fragment execution. Fragment execution could be interrupt by system in any time and fragment's transaction could be rollback on rerun. So fragment have to have the ability of restarting.

6) There are several requirements on input\output operations that caused by transactional nature of computations:

- the goal of input\output operation should be mapped on TVM value;
- input\output operations are executed by system core only;
- input\output operations will have really be executed only if their transaction is succeeded.

Main components of proposed architecture are presented on Fig.2.

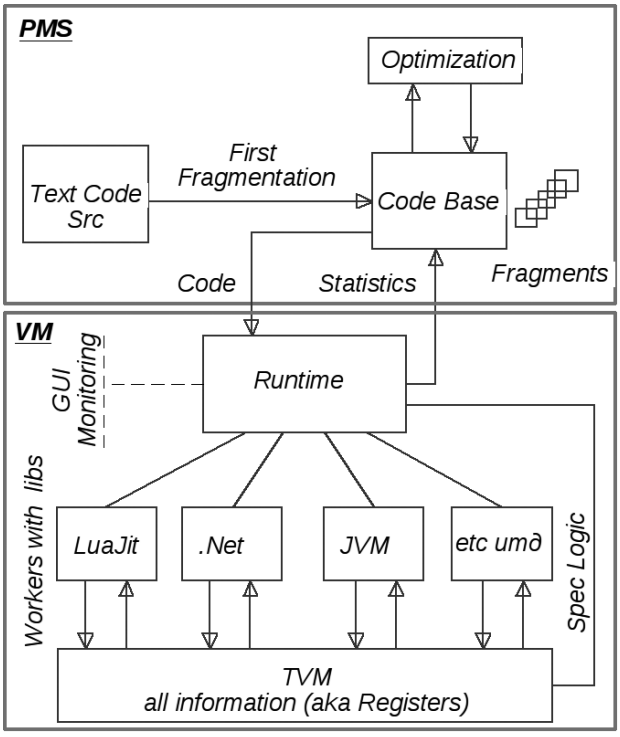

Fig. 2. Main components of proposed architecture

TVM component stores operational data and it's versions. The main function of TVM is monitoring the correct sequence of access to data. If there are violations of the access sequence, it reports the system. TVM could be based on any objective DB (e.g. Objectivity [10]) or document store like MongoDB [11]. The main difference of TVM from other data storage services is hierarchy of transactions function. TVM stores not only data, but fragments of program too.

Runtime component responsible to searching optimal node computation to fragment execution, to measurement of fragment execution properties (execution time, number of calls to TVM, permissions and so on). Runtime manages transaction lifecycle, input\output operation to TVM mapping and it is committed to asynchronous execution of all operations.

One could compare our runtime with such system as Mozart-Oz runtime [12], SEAM (Simple Extensible Abstract Machine) [13], Splicemachine (rdbms under hadoop with distributed computing) [14] and others. Our system differ from these runtimes by code execution statistic collecting and analyses and by intensive transaction using.

PMS (Polymorphism mutation system) component do program source code translation for code analyses and preliminary fragmentation. After the moment, when first statistic of program execution is collected by runtime PMS do fragment optimization and searching for optimal model of execution.

In contrast with other optimizing compilers PMS do optimization in runtime based on execution statistics. PMS could do code mutation to achieve different code fragmentation and compare results of execution.

There are two methods to automatic granularity selection of executable code and code fragmentation:

1) method of lazy and iterative fragmentation;

2) method of generation all version of fragments and subfragments and its compare.

Second method is a brute-force method, because we need recursively split source code to set of fragments and its' part and then build all combination. This is NP complex task and so we couldn't apply second method directly.

In our system we use first method.

To reduce number of fragmentation variants we use knowledge about fragments dependencies. This knowledge allows to exclude most meaningless and incorrect variants.

The optimization of granularity begins with full decomposition of source code into the fragments with some limited size (it is even possible to split source code to fragments with bytecode instruction).

To reduce number of test measurement of fragments execution properties we algorithm, that model result properties of union several fragments based on fragments dependencies and properties of each individual fragment execution.

PMS also could change fragmentation scheme based on system execution statistics. PMS represents source code and fragments in stacked parallel form. Such form helps PMS to analyze fragment dependencies and regenerate new set of fragments.

Example of such representation is shown on the Fig.3.

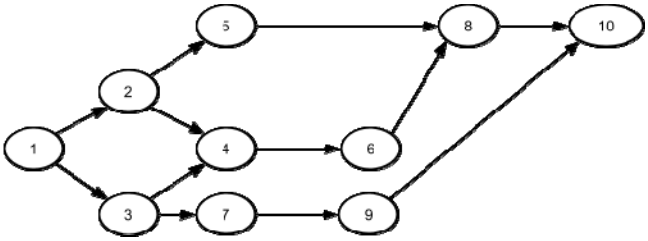

Fig. 3. Stacked parallel form of fragments set

In fact PMS try to solve such a task – which fragment need be combined to give best result.

Benefit from the combination is obtained by reducing transport costs. Then important factors are limitation of computation power of node hardware or reducing parallel degree of program caused by reducing number of fragments.

During optimization costs are reduced or compensated by concurrency of program execution.

If PMS continue to combine fragments to make bigger fragments that the moment happens when performance will suffer, because limitation of parallel execution is more important, then reducing transport cost.

So in such optimization process we have found optimal granularity level - the point where size of combined fragments is optimal for given data distribution, node hardware and network connection and parallel degree of algorithm (Fig.4).

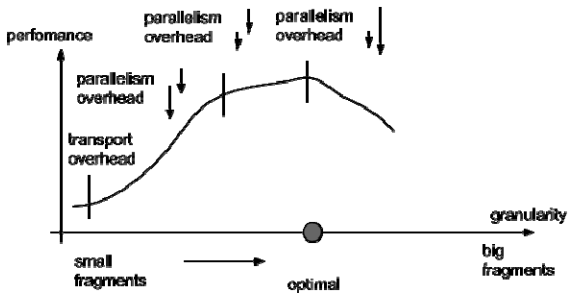

Fig. 4. Finding optimal granularity point

It should be noted, that in practices development of application with our VM and tool necessarily statistics of application execution will be collected during application testing. During first phases of testing performance of the application couldn't be important factor, so VM will have a time to optimize program structure and model.

Described approach reduces risk of changes in PS and PM that would be automatically happens during PMS optimization technique working.

IV. TVM EVALUATION

At the time of this paper writing we have first version of TVM implemented (another components are in development now). TVM is one of important component of VM because it holds all data access operation and could limit overall system perfomance. Apart this runtime and PMS need information about transaction execution.

Scheme of working with TVM is presented on Fig.5.

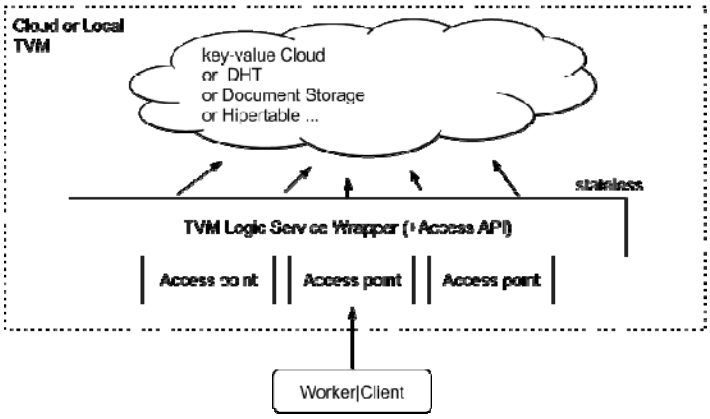

Fig. 5. Working with TVM

Simplified version of access API is presented below:

```
tvm_cell = getGlobal( acces_token,
[kind], [purpose] )
set(tvm_cell, tvm_value );
set(tvm_cell, index, tvm_value);
set(tvm_cell, string_key, tvm_value);
set(tvm_cell, number_key, tvm_value);
set(tvm_cell, boolkey_key, tvm_value);
set(tvm_cell, tvm_key, tvm_value);
set(tvm_cell, function_key, tvm_value);
set(tvm_cell, closure_key, tvm_value);
tvm_value = get(tvm_cell);
si = getisize(tvm_cell);
sf = getsize(tvm_cell);
tvm_value = get(tvm_cell, string_key);
tvm_value = get(tvm_cell, number_key);
tvm_value = get(tvm_cell, bool_key);
tvm_value = get(tvm_cell, tvm_key);
tvm_value = get(tvm_cell, function_key);
```

```
tvm_value = get(tvm_cell, closure_key);
subtransaction = call(tvm_cell, P)
subtransaction = call(tvm_value, P)
```

TVM cell is interface element that represents TVM storage element for user. Interesting that TVM value and even a key could be function and function closure.

To test TVM and evaluate its performance we implement such measurement scenario:

1) generate test code with access operation to random TVM cell within transaction of random structure;
2) run several iteration of test on local machine.

Example of test's code is presented below:

Algorithm 1 TVM testbench code

```
START_BENCH(startBench);
tvm.readFH("v5","tr1.49.32.29",tforrestart);
ELAPSED_TIME(startBench,"read v5");
START_BENCH(startBench);
tvm.writeFH("v9","tr9.17.14.15.79","val1497",tforrestart
);
ELAPSED_TIME(startBench,"write v9");
START_BENCH(startBench);
tvm.writeFH("v3","tr3.40","val416",tforrestart);
ELAPSED_TIME(startBench,"write v3");
… etc
```

"tr1.49.32.29" and other string with such pattern are transaction names.

We run each test 100 times with 100 operation in each test. Each test consists of 70% write operations and 30% read operations. We use leveldb [15] and in memory data structures as low level data storage upon which TVM works. Result for leveldb is represented on Fig.6.

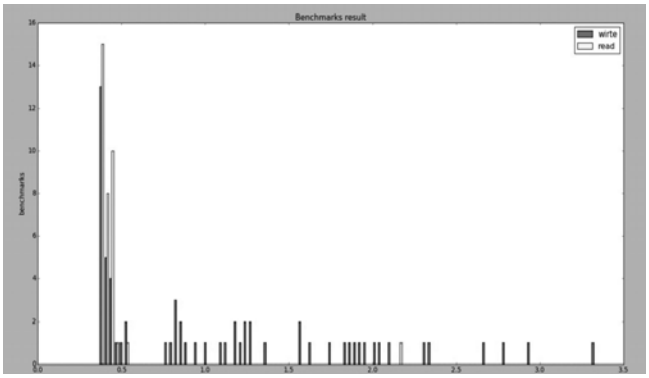

Fig. 6. Performance evaluation of TVM upon leveldb

When we evaluate performance of TVM based on in memory storage we have average access time equal to 10-20 microseconds or 100000 read\write operations per second. Such results are comparable with Redis storage [16].

These results show that our TVM implementation could be used in cloud services as transaction key-value data storage.

It should be noted that TVM even more useful not as a separate component, but a component of cloud VM – because information about transaction execution could provide useful knowledge to runtime and PMS component to reorganize program structure and program model and choose optimal code granularity for the case.

V. Conclusion

Described virtual machine could be used for solve BigData problems. Proposed architecture addresses both effective algorithm parallel implementation and data access and distribution aspects of every high load cloud service. One could argue, that such project requires significant resources , a large development team and infrastructure to support. We understand these and publish our work to collaborate with specialist who could be interested in solution like ours.

Proposed system should be used not only as data analyses tool, but also as an instrument for algorithm evaluation and generation its' new version capable work on scalable parallel cloud operation systems.

As a conclusion we could point out, that our implementation of virtual machine is in progress. We plan to present our fragmentation algorithm in more detail form and publish more samples of code transformation to illustrate main principles behind architecture.

We did not pay attention to such interesting aspects of our project as cross platform and multilanguage of VM (one could note that runtime use different VM to execute code fragments), also we didn't write about message passing and routing algorithm, which of course very important for any scalable cloud service. These are the points of most interest for us in near future.

Acknowledgment

This work was supported by the Ministry of Education and Science of the Russian Federation (No 14.607.21.0012 (RFMEFI60714X0012) agreement for a grant on 'Conducting applied research for the development of intelligent technology and software systems, navigation and control of mobile technical equipment using machine vision techniques and high-performance distributed computing').

REFERENCES

[1] P.B. Beskow, “Parallel programming models and run-time system support for interactive multimedia applications”, Web: http://heim.ifi.uio.no/~paalh/students/PaulBeskow-phd.pdf

[2] S. Marlow, “Parallel and Concurrent Programming in Haskell, Microsoft Research Ltd., Cambridge, U.K. 2013”, Web: http://community.haskell.org/~simonmar/par-tutorial.pdf

[3] D. Cordes, M. Engel, O. Neugebauer, P. Marwedel, “Automatic Extraction of pipeline parallelism for embedded heterogeneous multi-core platforms”, *in Proc. 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES '13),* Article 4, pp, 1-10.

[4] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, “SCOPE: easy and efficient parallel processing of massive data sets”, *in Proc. VLDB Endow. 1, 2*, August 2008, pp. 1265-1276.

[5] M. DeVuyst, D. M. Tullsen, and S. W. Kim, “Runtime parallelization of legacy code on a transactional memory system”, *in Proc. 6th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC '11),* pp. 127-136

[6] H. Sundell, A. Gidenstam, M. Papatriantafilou, and P. Tsigas, “A lock-free algorithm for concurrent bags”, *in Proc. twenty-third annual ACM symposium on Parallelism in algorithms and architectures (SPAA '11)*, pp. 335-344

[7] K. Malde, “Can Software Transactional Memory Make Concurrent Programs Simple and Safe?”, *in Proc. International Conference on Bioinformatics Models, Methods and Algorithms (BIOINFORMATICS 2013)*

[8] C. Prehofer, M. Zeller, "A hierarchical transaction concept for runtime adaptation in real-time, networked embedded systems", *in Proc. IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA)*, 17-21 Sept. 2012

[9] M. Bolingbroke, S.P. Jones, "Supercompilation by Evaluation", Web: http://research.microsoft.com/en-us/um/people/simonpj/papers/supercompilation/supercomp-by-eval.pdf

[10] Objectivity DB, Web: http://www.objectivity.com/

[11] Mongo-DB, Web: http://www.mongodb.org/

[12] L. Kornstaedt, "An Interoperability-based Implementation of a Functional Language on Top of a Relational Language", Electronic Notes in Theoretical Computer Science 59 No. 1, 2001.

[13] T. Brunklaus, L. Kornstaedt, “A Virtual Machine for Multi-Language Execution. Technical report, Programming Systems Lab”, Web: https://www.ps.uni-saarland.de/seam/

[14] Spice Machine, Web: http://www.splicemachine.com/

[15] LevelDB, Google, Web: http://leveldb.org

[16] J. L. Carlson Redis in Action, Web: http://www.manning.com/carlson/RiAch4sample.pdf