Model of distributed computations in virtual testbed

Ivan Gankevich
Saint-Petersburg State University
Saint-Petersburg, Russia
e-mail: igankevich@cc.spbu.ru

Alexander Degtyarev
Saint-Petersburg State University
Saint-Petersburg, Russia
e-mail: deg@csa.ru

ABSTRACT
Virtual testbed is a complex system of modeling natural and anthropogenic phenomenon and a nature of problems being solved is so demanding to computer resources that it requires efficient model of computations to be developed in order to complete experiments in time. Analyses have shown that such a model should provide resilience to node failures, ensure transactional behavior of computations and also be capable of both static and dynamic load balancing. The model has been developed on the basis of actor model which is analogous to hierarchical model governing work of a team with a large number of participants. The model was adapted to accommodate its usage in a distributed environment and research results can be used to embed it in a compiler that translates source code of a sequential program into parallel directives of the model.

Keywords
Virtual worker, distributed system, testbed.

1. INTRODUCTION
The notion of computation is not only inherent to machines but also for people and a process of solving a problem by a team of people can be divided into three divide and conqueror stages. During the first stage principal divides complex problem into smaller subtasks and assigns them to his subordinates, in the second stage subordinates solve the assigned tasks in parallel and in the last stage principal collects the results, approves them and then consolidates them into a single solution. Unsuccessful approvals lead to reiteration of the assigned tasks until the agreement is reached which in turn leads to asynchronous stage transitions among subordinates when principal waits for valid results from one person and collects the approved results from others. In summary, a process of solving a problem by a group of people is composed of three stages: factoring of a problem into subtasks for parallel execution and collecting computed results into a final solution - and transitions between stages occur asynchronously due to approval mechanism.

The way a group of people solves the problem can be compared to the way that problems are solved by a multiprocessor computer, however, it has its own peculiarities. First of all, a problem being solved is decomposed into two parts: data and code used to process it - and process of solving problem is represented by interaction between a data flow and a control flow. In such interpretation data goes through the steps of 1) decomposition into independent chunks and distribution between available memory banks, 2) local processing on each processor and 3) consolidation; in similar vein, code goes through the steps of 1) decomposition into distinct code parts which are distributed between processors, 2) local computations on each processor and 3) synchronization of all control streams. Data flow and control flow are reflected using queue and pipeline mechanism and this is how transition between solution stages is carried out. So, the way multiprocessor computer solves the problem is analogous to the way that the problem is solved by a team of people with exception to solution process being decomposed into data and control flow (Table 1).

	I	**II**	**III**
Stage	Division	Execution	Collection
Problem	into subtasks	of subtasks	of results
Data	into chunks	of transformations	of transformation results
Code	into threads	of threads	synchronization

Table 1. Comparison of the way a multiprocessor machine and a team of people solve a problem using divide and conqueror approach.

In a general case a problem being solved is so complex that the solution can be obtained not by one team but by a group of teams and their leaders composing single hierarchy, and such organization does not change the model but increases its complexity. Considering such organization the solution of subtasks also can be decomposed into sequential stages and it causes the work to flow down through hierarchy of workers and the results of this work to flow up through the same hierarchy. In addition to this, in a group of teams each principal can also be someone's subordinate, therefore, the failed approvals can cause reiteration of several layers of work. So, any work that flows down the hierarchy is always accompanied by the corresponding results flowing up the hierarchy. Finally, unpredictable or planned activities such as workers going to vacation or dismissal cause modification of a hierarchy and there should be a mechanism of reassigning a task to another worker instead of dismissed one and a mechanism of reelection of a new chief principal. To summarize, the process of solving a complex problem by a group of teams is characterized by a work flowing down the dynamically changing hierarchy of workers accompanied by the approved results flowing in the opposite direction.

In a way that teams can be composed into single hierarchy of workers to solve complex problems multiprocessor machines can be connected to form a single cluster and such cluster of computers has a similar principle of operation. Much like a work flows down a hierarchy of workers data and code flows down a hierarchy of computers (a network) and results of computations flow up the same hierarchy in both cases (Figure 1). Analogous to hierarchy of workers being changed over time by adding and removing people from teams a network topology is changed by adding and removing computers due to hardware failures or withdrawal and it causes work to be reiterated on different set of machines. So, given that a network topology of a computer cluster can be represented by hierarchy of machines its principle of operation is analogous to that of a hierarchy of workers and can be described as a generalization of operation of a single multiprocessor machine.

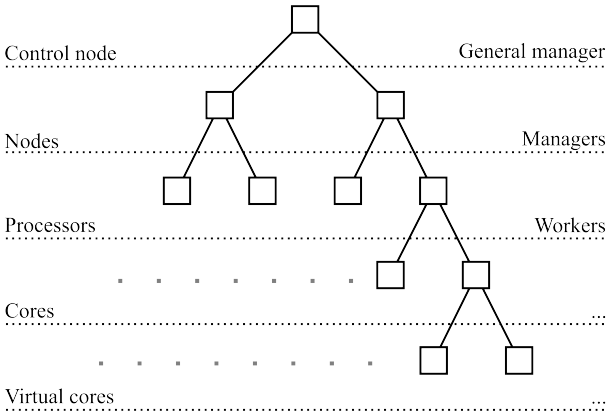

Figure 1. Mapping of hierarchy of virtual workers to hierarchy of computers.

All in all, computing model of a computer cluster is similar to how a group of teams solves the complex problem together and computing model of a single machine is similar to a process of solving problem by a single team of workers. These models can be combined by matching staging principle of doing work with a mechanism of interaction between separate cluster nodes. This is how the generalized model governing computations in distributed environment can be obtained and one possible way of development of such model is described in the paper.

2. DISTRIBUTED COMPUTING MODELS COMPARISON

There are several standards and experimental technologies governing computations in distributed environment and there are typical techniques of organizing distributed computing used several standards at once. These are compatibility with different parallel computer architectures and capability to work not only with distributed data but also with parallel computations (MIMD in Flynn classification [12]). In addition to this, there are some properties that are important in distributed environment; these are fault tolerance which importance is increased with a size of a computer cluster and load balance which is a requirement in heterogeneous network of computers. All in all, these techniques and properties can be used to access evolution of technologies governing computations in distributed environment (Table 2).

Although, data and code often considered separately, techniques to work with them in distributed environment are analogous to each other and can be used together. In case of code parallelism means a possibility to decompose control flow into distinct streams solving separate subtasks and pipelining means a possibility to sample each of the streams into distinct sequential parts. In similar vein, in case of data distribution means a possibility to divide data array into distinct chunks and queuing means possibility to process data in a defined order. Finally, in either case recursion means a possibility of recursive application of the described techniques so that they are orthogonal to each other. To summarize, three general techniques to work with code and data in distributed environment complement each other and can be used together to factor initial data structure of a given problem into separate and connected parts and to factor a process of its solution into parallel and sequential stages (Figure 2).

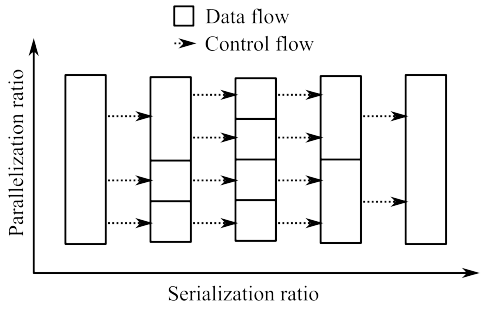

Figure 2. Interviewing of data and control flow: rectangles represent different data structures and arrows represent different transformations applying to chunks of data; parallelization ratio measures an ability to efficiently distribute computation of a single workload and serializaiton ratio measures an ability to efficiently distribute processing of multiple workloads (a number of units in a pipeline).

Much like data and code has different techniques to work with them in distributed environment fault tolerance and load balance can also be accomplished in two different ways complementing each other. The first way represents "integral" approach and is based on usage of additional redundant machines to tolerate hardware failures and usage of virtual machines with controlled parameters to balance load. This approach is used to develop fault tolerant distributed databases on the basis of distributed hash tables [3, 4]. The second way represents "differential" approach and is based on factoring workload into sufficiently small pieces that are distributed among available machines and can migrate between them. Such approach is used to process and to store in a consistent way big volumes of data [1, 2]. Since, the first way is purely technical and the second way is programmatic they can be combined to leverage efficiency.

	Standards				Experimental technologies			
	1	2	3	4	5	6	7	8
Architecture								
SMP	+	+	+	+	+	+	+	+
PVP			+				+	+
MPP		+		+	+	+		+
Data								
Distribution	+	+	+	+	+	+	+	+
Recursive dist.	+					+		
Queuing			+	+				+
Code								
Parallelism	+	+	+		+	+		+
Recursive par.	+							
Pipelining			+	+				+
Other prop.								
Fault tolerance				+				
Load balance	+			+	+	+		

Table 2. Comparison of standards and experimental technologies governing computations in a distributed environment.. 1 – OpenMP, 2 – MPI, 3 – OpenCL, 4 – MapReduce, 5 – MPC [13], 6 – STAPL [14], 7 – OpenACC, 8 – VirtualCL.

Comparison of standards and experimental technologies using distributed techniques shows that evolution of technologies goes towards load balancing and support for vector and cluster architectures (Table 2) and current standards lack support for fault tolerance and recursive techniques, however, implementation of standards are not always identical. For example, OpenMP defines directives for recursive data distribution (nested #omp parallel for

loops) but neither implementations of this standard fully support such directive nor the support maps it efficiently to the underlying hardware. Moreover, this directive is rarely combined with recursive parallelism directive (#omp task). Another example is OpenCL standard which aimed to support both SMP and PVP architectures but code optimization techniques use different criteria for CPU and GPU devices [10]. In summary, peculiarities in implementations of standards lead not only to the lack of efficient support for recursive parallelism and data distribution but also for inefficient support for PVP architectures.

All in all, standard and technologies of distributed computing use similar techniques to manage computations in distributed environment, however, not even a single standard implement these techniques in full. Data distribution and parallelism techniques are realized separately from each other and do not always allow recursive application (arbitrary nesting). Finally, load balance and fault tolerance are implemented mostly in a programmatic way.

3. GENERALIZED DISTRIBUTED COMPUTING MODEL

Analysis of distributed computing models shows that each concurrency technique operates on some sort of parts of a whole problem and those parts are analogous to each other in terms of their behavior. In case of data these parts represent data structures that can be recursively divided into smaller chunks to put into separate processing queues. In case of code those parts represent code sections that can be divided into smaller ones to form parallel pipelines. So, in both cases the notion of part is connected with some volume of code or data.

In contrast to concurrency techniques computation properties such as fault tolerance and load balance represent not a behavior of parts of a problem but a behavior of a process of finding solution as a whole. So, the rule of fault tolerance in case of data ensures that in event of failure of a machine storing some volume of data you can always find another machine that stores the same volume of data and in case of code it ensures that you can always find another machine to execute the code one more time. Load balance rule ensures that each machine in a cluster is loaded as much as it is allowed by available computational power and operating memory capacity. In summary, for the system to adhere to both of these rules data replication and code migration should take into account workload of the system.

Described techniques and rules can be consolidated in a single entity by introducing a notion of a virtual worker which has a set of distinct properties and generalized behavior. First, virtual worker consists of both the data and the code processing it and such an approach unifies a notion of a data and control flow. Second, each virtual worker is capable of spawning its own virtual subordinates with their own data and code structure and also is able to approve the results of their tasks. Approval of results means possibility to commit or rollback completed work and describes transactional behavior of a virtual worker. Third, virtual worker is mobile and can be migrated to another machine when current one is overloaded. All in all, a virtual worker is

- a unit of parallelism representing independent fractions of a problem,
- a unit of hierarchy controlling execution of tasks of his subordinates,
- a unit of transaction capable of committing and rolling back actions of his subordinates and
- a mobile unit supporting migration between different computing nodes of a cluster.

Computational process involving virtual workers is composed of three divide and conqueror stages and has a straightforward mapping to a computer system architecture. The process starts by spawning main virtual worker and the first stage consists of recursive spawning of subordinates that are put to a processing queue. Then the second stage consists of direct processing of virtual workers and the third stage consists of collection and combination of results that flow up the virtual workers hierarchy. The mapping of the process occurs by matching virtual workers hierarchy with hierarchy of machines in a computer system and processing queues are created for each machine in a system. So, much like in a hierarchy of real workers computational process is composed of three divide and conqueror stages and by matching this hierarchy to a topology of a computer network this process can be mapped to a system architecture.

In a system of virtual workers a failure of a node executing main virtual worker can cause abnormal termination of a whole program and to ensure its fault tolerance a distinct approach should be used. In contrast to subordinates fault tolerance being provided by execution of them on different nodes in case of a main virtual worker failure his closest subordinates should reelect him using the algorithm of distributed consensus. Such an algorithm is not completely reliable [5] and its use during subordinates fault seems unpractical, however, in case of a main virtual worker it is the only way of restoring system to a healthy state.

In summary, a solution of a problem using computer system involves processing of data and control flows and these flows can be combined into a single sampled stream. Each sample of this stream represents a single mobile entity - a virtual worker composed of data and code and capable of migrating between machines of a system. Fault tolerance of virtual workers is provided by rerunning failed tasks on different machines in case of subordinates and by means of reelection a new principal using distributed consensus algorithm in case of a main virtual worker.

4. COMPARISON TO EXISTING APPROACHES

Generalized computational model can be compared to the existing approaches of distributed computations to show its advantages and disadvantages and among all these approaches there are two closest ones. The first approach was an attempt to solve the problem of distributed computations on the basis of computer system architecture implementing so called macro pipelining processing [6]. The second approach is based on programming language implementing actors model of distributed computations [7] and was a research in a field of artificial intelligence. Other approaches are based on an object-oriented and functional programming languages and each of them has its own advantages and disadvantages compared to generalized model.

Actors model was designed to describe interactions of agent composing artificial intelligence systems and a notion of an actor has much in common with a notion of a virtual worker but there are also some discrepancies. First of all, each actor upon receiving a message can react in one of the three possible ways: create more actors, send messages to other actors and change its state and reaction to the next message whereas in generalized model there is no notion of a message and a process of spawning subordinate workers is combined with a process of sending message. Moreover, such an approach is considered uniform: “the most exciting of such models, and the one using the greatest uniformity of construction, is one in which the communications are

themselves actors" [8]. Second, actors model does not provide the governing direction of interactions between actors that leads to problem solution whereas the generalized model uses hierarchy of virtual workers to ensure convergence: every subordinate always reports results of his work to his principal which ensures reaching one goal by all virtual workers. Other properties of actors model similar to the ones generalized model has: each model describes arbitrary configured dynamic graphs of actors (virtual workers), provides incremental synchronization by means of a concurrent queues (synchronization occurs only between communicating entities and not between all the entities at once) and describes computations with no restriction to a certain system architecture. To summarize, generalized model and actors model are similar in a way they represent computation by interaction of unified entities and dissimilar in ways of organizing these entities to solve a problem and also in implementation of entity communications.

Much like a theoretical model a programming language also represents a model of computations that represents problem solution by composition of interacting objects or functions and a notion of an object and function can be related to a notion of a virtual worker. In object-oriented programming languages problem solution is accomplished by defining main control object that creates subordinate objects, connects them in a single hierarchy and launches their methods; this process is analogous to building of virtual workers hierarchy and putting them in a processing queue. In functional languages problem solution is represented by composition of functions and monads that are executed as a single pipeline which is analogous to virtual workers queues. Moreover, some functional languages already use hierarchy to facilitate fault tolerance (Erlang supervisory trees) [11]. In both programming paradigms object and function are composed from data and code (each function has a copy of its arguments) and that makes them independent and, given appropriate implementation, capable of migrating from one machine to another. So, basic building blocks of object-oriented or functional program are similar in terms of structure and principle of operation to a notion of a virtual worker.

Traditionally, modern compilers can automatically find data parallel regions of code [9] and integration of a generalized model with compiler can broaden possibilities of finding independent code regions and facilitate not only parallel but also pipelined workloads.

Advantages and disadvantages of a generalized model can be summarized as follows.

Advantages

- joint data and code migration
- incremental synchronization
- transparent transaction mechanics
- dynamic load balancing
- unified and simple building block of a program
- single source code for MPP and SMP architectures

Disadvantages

- usage of distributed consensus algorithm for main virtual worker reelection
- inability to define optimal volume of data and code to compose a single virtual worker

5. POSSIBLE APPLICATIONS

Possible application areas of the proposed model are twofold and it can be used both as a base model of middleware toolkit or as a primary model of computation in a software application. Although, the first use case has not been discovered yet, the model has found its natural fit in scientific source code simulating multidimensional physical phenomena (virtual testbed). Virtual testbed is a complex system modeling simultaneous natural processes, their interactions with each other and their interactions with a dynamic object such as a ship in rough seas. Experience in developing such numerical simulations shows that virtual workers approach allows to factor initial problem to several loosely-coupled components and tie them together in a unified parallel way. In addition to this, this approach shows comparable performance to conventional OpenMP directives. So, the generalized model of computation can be used in numerical modeling software and possibility of its usage in distributed environment and its usage as a base of middleware toolkits requires further investigation.

6. CONCLUSION

Generalized computation model describes one possible approach of developing distributed applications which takes into account deficiencies of existing concurrency techniques and takes one step further towards implementing transparent load balancing, providing fault tolerance and transaction mechanisms. Given appropriate realization the model can be used to create applications adapting to computer system architecture, with configurable fault tolerant and load balancing mechanisms and with a source code composed from unified virtual workers specifications simplifying development of programs.

ACKNOWLEDGEMENT

The research was carried out using computational resources of Resource Center Computer Center of Saint-Petersburg State University (T-EDGE96 HPC-0011828-001) and supported by Russian Foundation for Basic Research (project N 13-07-00747) and St. Petersburg State University (project N 9.38.674.2013).

REFERENCES

[1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[2] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.

[3] DeCandia, Giuseppe, et al. "Dynamo: amazon's highly available key-value store." ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007.

[4] Lakshman, Avinash, and Prashant Malik. "Cassandra—A decentralized structured storage system." Operating systems review 44.2 (2010): 35.

[5] Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." Journal of the ACM (JACM) 32.2 (1985): 374-382.

[6] V.M. Glushkov. Introduction to ASU. Kiev: Tehnika, 1974 (in russian)

[7] Hewitt, Carl, Peter Bishop, and Richard Steiger. "A universal modular actor formalism for artificial intelligence." Proceedings of the 3rd international joint conference on Artificial intelligence. Morgan Kaufmann Publishers Inc., 1973.

[8] Agha, Gul Abdulnabi. "Actors: a model of concurrent computation in distributed systems." (1985).
[9] A.V. Bogdanov, I.G. Gankevich Practical Efficiency of Optimizing Compilers in Parallel Scientific Applications // Distributed Computing and Grid-Technologies in Science and Education. Proceedings of 5th International Conference — Dubna, — 2012
[10] Degtyarev A., Gankevich I. Efficiency Comparison of Wave Surface Generation Using OpenCL, OpenMP and MPI // Proceedings of 8th International Conference «Computer Science & Information Technologies» — Yerevan, Armenia, — 2011. — P. 248-251
[11] Armstrong, Joe. Making reliable distributed systems in the presence of software errors. Diss. KTH, 2003.
[12] Flynn, M. (1972). "Some Computer Organizations and Their Effectiveness". IEEE Trans. Comput. C-21: 948.
[13] Pérache, Marc, Hervé Jourdren, and Raymond Namyst. "MPC: A unified parallel runtime for clusters of NUMA machines." Euro-Par 2008–Parallel Processing (2008): 78-88.
[14] An, Ping, et al. "STAPL: An adaptive, generic parallel C++ library." Languages and Compilers for Parallel Computing (2003): 195-210.
[15] Barak, Amnon, and Amnon Shiloh. "The Virtual OpenCL (VCL) Cluster Platform."