Distributed Block Sort: a Sample Application for Data Processing in Mobile ad HOC Networks

S.N. Popov, I.V. Kazakova, S.V. Vostokin

Abstract. The study focuses on the design methodology for network applications with centralized control suitable for the mobile ad hoc networks. The idea of the methodology is to use actor systems and microservice approach to generate a stream of tasks that must be executed by devices connected to the mobile ad hoc networks. The article presents step-by-step solution of distributed block sorting problem to illustrate the design methodology. The graphical notation of actor-based microservices and definition of its execution semantics were given. The notation was used to compose the design diagram of the block sorting application for distributed data processing. After that, we discussed tools and techniques for getting the C++ source code of the block sorting application from the design dia-gram. Finally, we presented results of experimental research of the block sorting application throughput in a multi-threaded execution environment. It was found that the methodology can be effectively used to perform various operations using hardware resources associated with ad hoc mobile networks.

Keywords: Microservice Application, Mobile Ad Hoc Network, Distributed Da-ta Processing, Actors, Parallel Programming, OpenMP, Block Sort.

I. INTRODUCTION

While solving many scientific and technical problems, it becomes necessary to pro-cess large data sets. The capabilities of serial applications make it impossible to man-age the hardware resources of modern computing systems effectively, so the use of parallel and distributed programming methods is of great interest among scientists and software developers. There is a possibility of centralized processing in data cen-ters, but it is not always effective. Performing calculations on any client devices con-nected to the network is another option for resolving this issue. However, this ap-proach poses the problem of writing applications that use the maximum amount of mobile ad hoc network resources [1]. In the article we propose a design methodology for network applications with centralized control.

II. METOD

This article considers a method of organizing calculations using the actor model by an example of distributed block sorting. The main elements of the model are actors - active agents, performing certain actions according to the specified scenario [2,3]. Actors were used to build a microservice control application. The main advantage of this style of development is the possibility of such an organization of the application, in which each of its component services performs part of the tasks, abstracting from others, and the interaction between elements uses standard protocols [4].

There are two major steps in the application design. The first (design) step is the build-ing of problem decomposition using microservice. The second (implementation) step is the definition of a behavior for each type of microservice and for each type of interface.

We use a special graphical notation of microservice and definition of its execution semantics. The microservice entity is depicted as a jigsaw puzzle tile (see Fig.1). The concave part of the tile denotes a service port of the microservice, while convex part of the tile denotes a client port. Each microservice may have both server and client ports. One server port may be bound with one or many client ports, while one client port may only be bound with one server port. The client-server interaction is bidirec-tional. That means sending an initial request from client to server and getting a re-sponse back from server to client. Each message (request or respond) may carry some user defined data or may be an empty message. Both server and client port have an interface type. One can connect ports of the same type only.

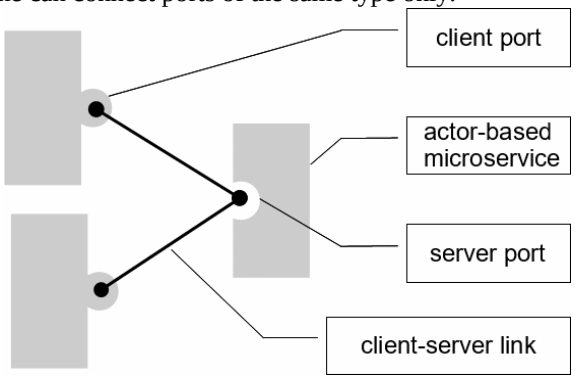

Fig. 1. The graphical notation of actor-based microservices.

Our microservices implement actor execution semantics. Each service is passive and is waiting for incoming messages (requests or responds) to be activated. Only one message can be processed by a service at a time. If several messages are sent to a microservice (request or respond), they are processed one by one in the serial order. As a result, we have to define initially active services to start the program execution. Also there are two ways of finishing the execution. The execution may degenerate if, at some moment, there are no active messages. To keep the

Revised Manuscript Received on May 28, 2019.

S.N. Popov, Samara National Research University 34, Moskovskoye shosse, Samara, 443086, Russia.
I.V. Kazakova, Samara National Research University 34, Moskovskoye shosse, Samara, 443086, Russia.
S.V. Vostokin, Samara National Research University 34, Moskovskoye shosse, Samara, 443086, Russia.

execution, a microservice should produce new messages while processing old ones. The normal way of finishing the execution is an explicit notification of completion. It occurs then some microservice calls the completion command.

In the article we apply the microservice decomposition to the following sorting algorithm.

```
for (int i = 0; i<N; i++)
   block_sort(i);
for (int i = 1; i<N; i++)
   for (int j = 0; j<i; j++)
      block_merge(j, i);
```

Here we sort N blocks of M integers using the two predefined block operations. The block_sort(i) operation sorts the i-th block of integers from the initial array of M integers with Hoare's quick sort algorithm. The block_merge(j,i) operation merges the j-th and the i-th presorted blocks. As a result the concatenation (j,i) of the two blocks becomes ordered. The idea of the microservice decomposition is to get an application that can generate a sequence of independent block_sort / block_merge tasks to be executed by peers in the ad hoc network.

We developed an application with the following structure (see Fig. 2).

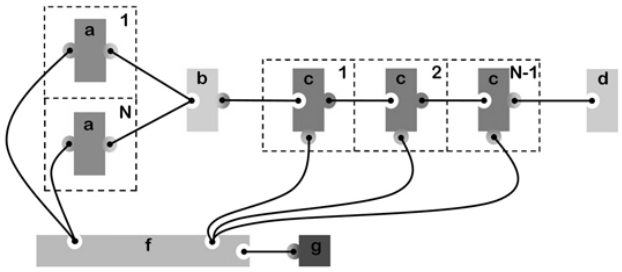

Fig. 2. The structure of the block sorting application for distributed data processing.

Let us consider in detail the elements of the structure shown in the Figure 2, and the scheme of their interaction. The system can be divided into two subsystems, namely the management subsystem, which includes the actors named sorter (a), producer (b), merger (c), stopper (d), and the execution subsystem, including the actors named everest (f) and timer (g).

Suppose that initially there is an array of data consisting of M integers, which must be sorted. We divided it into N equal blocks. An actor of the sorter type (a) is created on each block. The sorter (a) interacts with producer (b) and everest (f) actors.

An important element in this system is the everest (f) actor. In the future, it will interact with the Everest service [5], which, in turn, will perform the received tasks using the available hardware resources of the ad hoc network, and return the result to the system. The everest (f) actor is a proxy server whose main function is to reconcile the formats of requests received from the actors named sorter (a) and merger (c), and the responses received from the Everest service. At the moment the service uses OpenMP 3.0 task directive, which allow us to simulate the operation of the service on a multiprocessor system with shared memory. In addition to the sorter (a) and the merger (c), the everest actor (f) interacts with the timer (g) actor. Its main task is to perform a periodic launch of the everest (f) actor. If there are queued tasks in a process of the launch, then they are sent for execution. If there are completed tasks, then their results are returned to the senders.

After getting a result from the everest (f), the sorter (a) passes the execution message to the producer (b) actor. It, in turn, having received a sufficient number of notifications, begins to transfer them one by one into the chain of merger actors (c) in ascending order of block numbers. Due to the interaction of the merger (c) and the everest (f) actors, the blocks are merged according to the insert sorting method, after that a message is sent to the stopper (d), which stops the calculations.

To proceed from the first (design) step of the application development to the second (implementation) step we use the Templet Web framework and web-based integrated development environment [6,7]. This IDE automates (a) application code generation and (b) application deployment. For example, to get a skeleton code for the producer (b) microservice she/he types

```
#pragma templet *producer(in?mes,out!mes)
```

directive in the web-browser and gives a 'generate' command. The desired code for the microservice will appear automatically. The generated microservice can listen to the messages of type 'mes' on 'in' server port and can send the messages of type 'mes' from 'out' client port. All you need now is to define the message handles and data fields for the producer microservice.

Each microservice is a C++ class. To obtain the resulting application we construct entities of the microservice classes and link them together as shown below.

```
everest an_everest(e);
timer a_timer(e);
an_everest.timer(a_timer.p);
producer a_producer(e);
stopper a_stoper(e);
sorter** a_sorter = new sorter*[NUM_BLOCKS];
for (int i = 0; i < NUM_BLOCKS; i++) {
   a_sorter[i] = new sorter(e);
   a_sorter[i]->i = i;
   a_producer.in(a_sorter[i]->out);
   an_everest.s(a_sorter[i]->e);
}
merger** a_merger = new merger*[NUM_BLOCKS-1];
for(int i=0;i<NUM_BLOCKS-1;i++){
   a_merger[i]=new merger(e);
   an_everest.m(a_merger[i]->e);
}
mes* prev=&a_producer.out;
for(int i=0;i<NUM_BLOCKS -1;i++){
   a_merger[i]->in(*prev);
   prev=&(a_merger[i]->out);
}
a_stoper.in(*prev);
```

III. RESULTS

An experimental study of the correctness of operation

and possible acceleration of computations in the developed application was carried out (see Table I).

Table I. Execution time and acceleration of the block sorting application.

Number of blocks	Block size, MB	Sequential sorting algorithm (t_s), s	Sequential block sorting algorithm (t_b), s	Parallel block sorting algorithm (t_p), s	Acceleration (t_s/t_p)
2	64	9,77	10,00	6,31	1,55
4	32	9,69	10,59	4,43	2,19
8	16	9,70	11,80	4,27	2,27
16	8	9,63	13,99	6,90	1,40
32	4	9,46	18,59	9,88	0,96
64	2	9,63	27,26	19,26	0,50
128	1	9,67	43,80	36,86	0,26

We represent the data from Table I as a graph (see Fig. 3).

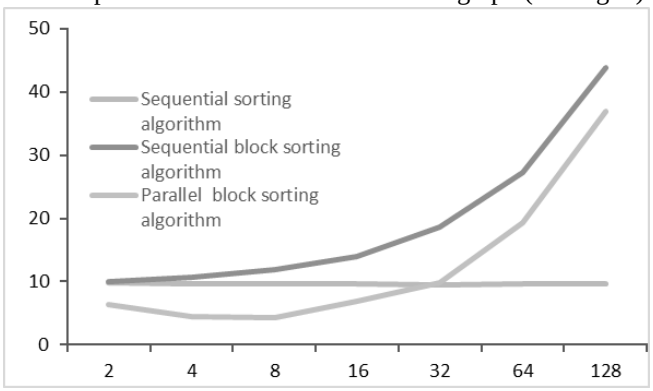

Fig. 3. Execution time of block sorting application.

The number of tasks sent to the network by the application for remote execution can be seen in the Table II.

Table II. Number of tasks to be sent for remote execution by the block sorting application.

Number of blocks	Number of merge tasks	Number of sort tasks	Total tasks
2	1	2	3
4	6	4	10
8	28	8	36
16	120	16	136
32	496	32	528
64	2016	64	2080
128	8128	128	8256

Sorting was performed on a multi-core system with shared memory with Intel Core i7-3630QM 2.40 GHz processor. The experiment for sorting an array of 128,000,000 numbers demonstrated an acceleration of 2.27 times on a quad-core system supporting 8 hardware streams. We confirmed that it is possible to form a large number of tasks, and this does not lead to the degradation of the system throughput. In the future, it is planned to conduct a block sorting experiment with a real connection to the Everest service, using a network of personal workstations as the executor of block sorting and merging operations.

IV. DISCUSSION

The service composition is one of important elements of ubiquitous systems design [8]. Today Akka framework is a popular actor-based tool to implement the service composition [9]. However, Akka is efficient on dedicated cluster systems while the goal of our research is to manage large scale distributed systems.

During the planning of the application, we analyzed systems adapted for large scale execution, in particular, the software platform named BOINC [10]. BOINC is a soft-ware package consisting of a server and client part.

The server part is a set of functions for overall project management: registration of participants, distribution of tasks for processing, obtaining results, managing project databases. In our interpretation, this part is implemented using the interaction of the actor-based application and the Everest [5] distributed computing service, which makes task management more flexible. Also, using the actor model in the task man-agement program makes the process of writing applications more transparent and convenient. The similar results, but with a different actor implementation were ob-tained in [11].

The client part in the BOINC system is a universal client for working with various distributed computing tasks. Analogues in our system are agents created on different platforms and performing the role of resources in mobile ad hoc networks.

V. CONCLUSION

We developed an application that can be executed not only on a system with shared memory, but it can also be used to control massive computations in distributed envi-ronment. The design methodology for network applications development is suitable to perform operations that require significant hardware resources, to collect infor-mation from sensors, and to do other information processing task in the ad hoc net-works. Applications created in accordance with the methodology can reduce the cost of computations by more efficient use of existing network infrastructure.

ACKNOWLEDGMENT

This work is partially supported by the Ministry of Education and Science of the Rus-sian Federation in the framework of the State Assignments Program (#9.1616.2017/4.6).

REFERENCES

1. Boukerche, A., *Algorithms and Protocols for Wireless Sensor Networks*, Wiley-IEEE Press, 2008, ch. 1.
2. Hewitt, C, Bishop, P. and Steiger, R., "A Universal Modular Actor Formalism for Artificial Intelligence", *Proceedings of the 1973 International Joint Conference on Artificial Intelligence*, 1973, pp. 235–246.
3. Agha, G.A., *ACTORS: A Model of Concurrent Computation in Distributed Systems.* MIT Press Cambridge, MA, USA, 1986.
4. Martin Flower at al. Microservices. Available: https://martinfowler.com/articles/microservices.html.
5. Sukhoroslov O., Volkov S., Afanasiev A., "A Web-Based Platform for Publication and Distributed Execution of Computing", *Proceedings - IEEE 14th International Symposium on Parallel and Distributed Computing*, 2015, pp. 175-184.
6. Templet on GitHub. Available: https://github.com/Templet-language/newtemplet/.
7. Vostokin, S., Artamonov, Y., Tsarev, D., "Templet Web: the use of volunteer computing approach in PaaS-style cloud", *Open Engineering*, vol. 8, issue 1, 2018, pp. 50-56.

8. Machado, C.A., Silva, E., Batista, T., Leite, J., Nakagawa, E., "Architectural Elements of Ubiquitous Systems: A Systematic Review", *ICSEA 2013: The Eighth International Conference on Software Engineering Advances*, Venice, Italy, 2013, pp. 208–213.
9. Akka. Available: https://akka.io/.
10. Yi S., Kondo D., Anderson D.P., "Toward Real-Time, Many-Task Applications on Large Distributed Systems", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6271 lncs, part 1, Springer, Berlin, Heidelberg, 2010, pp. 355-366.
11. Guidi C., Lanese I., Mazzara M., Montesi F., "Microservices: a language-based approach", *Present and Ulterior Software Engineering*, Springer, Cham, 2017, pp. 217-225.

AUTHORS PROFILE

Stefan Nikolaevich Popov, undergraduate student of the Samara National Research University named after academician S.P. Korolev, Department of Information Systems and Technologies. The subject of his graduation work is building distributed application for sorting large data sets based on service-oriented architecture. His scientific interests includes parallel, distributed, and network programming, data science. During his last year in Samara University he also got a practice as part-time Junior Developer at Samara office of CQG, Inc., US-based company creating financial software solutions for market analysis. He wish to continue his education at Innopolis University, Kazan, Russia. He published two scientific papers. E-mail: stefan.pk@yandex.ru.

Irina Vladimirovna Kazakova, Engineer in the field of automated information processing systems and control. She works as an engineer at Joint Stock Company Space Rocket Center Progress and she is also a postgraduate student at Samara National Research University named after academician S.P. Korolev, Department of Information Systems and Technologies. The subject of her thesis was modeling of manufacturing processes in order to increase efficiency of use of hardware resources and output quality at the industrial enterprises. Now she is caring out her research in the area of parallel and distributed computing, including programming applications for mobile networks. She published 5 scientific papers in Russian and English, and she plans to protect her dissertation in the Scientific Council of Samara National Research University to get a PhD degree in the field of Computer Science. E-mail: ikazakova90@gmail.com.

Sergei Vladimirovich Vostokin, Doctor of Science (Tech.). He works as Professor at Samara National Research University named after academician S.P. Korolev, Department of Information Systems and Technologies. His doctoral dissertation research was focused on graphical object model of parallel processes and its applications in the field of numerical modeling. He led two PhD and ten Master students who successfully defended their dissertations. He is a member of international public organization "Academy of Navigation and Motion Control". His research interests includes parallel and distributed algorithms in numerical modeling, the programming automation, formal models of computations and their application for describing algorithms for distributed and parallel computing and control systems, computing in mobile networks. He is the author of more than 50 scientific papers in Russian and English including textbooks. E-mail: easts@mail.ru.