Data Protection in the Organization of Numerical Experiments in Distributed Computing [1]

Ilya Kurochkin, and Daniil Popov

Institute for Information Transmission Problems of Russian Academy of Sciences, Moscow, Russia

kurochkin@iitp.ru

Abstract. An approach to improve the efficiency of a distributed computing system is proposed. This approach involves reducing the number of initial replication copies by validating the results of one copy calculations. A method for verifying data integrity using cryptographic hash functions is described. And its adaptation for desktop grid based on BOINC platform.

Keywords: Distributed computing, Desktop grid, Replication copies, BOINC, Data protection, Cryptographic hash function.

1 Introduction

Distributed computing is one of the ways to solve complex computational problems. This method had a good performance at organizing large computational experiments in the interests of scientific groups and laboratories. The use of distributed computing systems to solve scientific problems may become the alternative to the use of supercomputers. As the need for large computation far exceed the capacities of available supercomputers and clusters.

Grid systems of personal computers (desktop grid) are a common tool for organizing large scientific experiments. There are a number of platforms (BOINC, HTCondor, Legion, Globus) to organize the calculations in desktop grids. Of these, the most popular is BOINC.

BOINC is an open software platform that allows us to use distributed computing to solve large computational experiments [1]. Such experiments are referred to as distributed computing projects. BOINC has client-server architecture and allows to connect not only personal computers but also servers, mobile devices, clusters to desktop grids.

At the moment, there are dozens of different scientific projects on the BOINC platform. For example, modelling of the charged-particle beam behavior at various parameters of the LHC@home accelerator control magnets impact on them[2]; Einstein@home project [3] deals with the search for spinning neutron stars (also called

[1] This work was funded by Russian Science Foundation (№16-11-10352)

pulsars) using data from LIGO and GEO gravitational wave detectors, as well as the Arecibo radio observatory.

2 Desktop grid features

Desktop grids have a number of features that should be taken into consideration when organizing the calculations:

- Heterogeneity of the distributed computing system nodes;
- Unreliability of connections and possible shutdown of computational nodes;
- Autonomy of the calculations on different nodes and impossibility of calculation coordination between nodes;
- Irregular time of continuous node operation;
- Errors or delays in the calculations.

If we analyze these features, we can formulate few stages to organize numerical experiments on a distributed system:

- The whole numerical experiment should be divided into small independent tasks;
- Computational complexity of one of the tasks should be small, within several hours calculation on an average personal computer;
- Each task should be calculated on several independent computing nodes (replication copies);
- Increase in the number of copies increases the reliability of correct calculation of an individual task and increases the speed of obtaining a correct result (on average for the entire set of tasks) but reduces the computational ability of the entire system;
- It is necessary to develop a calculations checking system.

2.1 Splitting into independent tasks

It is assumed that the original problem can be split into a large number of parts, which are many times greater than the number of computational nodes in a distributed system.

There is a certain class of scientific problems, which can be split into many independent tasks. The calculation algorithm remains unchanged, and only the input data change. This approach for the split of the original computational problem is called a data split or "bag of tasks" [4].

Examples of this class of problems can be the problems of simulation modelling, SAT problem [8], complete enumeration problems [9], problems of combinatorics [10] and etc.

2.2 Complexity of the tasks

It is necessary to choose the right computational complexity of the task. On the one hand, make it possible to solve the task completely on a regularly shut down computational node (for example, the node switch only in working hours). On the other hand, to make it difficult enough to reduce the share of overhead costs (transfer time of input data and results, unpacking and recording data, etc.). Typically, for distributed systems the task execution time on the compute node should be in the range from several minutes to 6 hours.

There are exceptions when one task runtime can be more than 10 hours or several days. But in this case, it is necessary to ensure regular saving of intermediate results.

2.3 Replication copies

To increase the likelihood of successful execution of the task in terms of a distributed system compute nodes possible shutdown, it is necessary to send out multiple copies of the same tasks to different nodes. If for each task, n copies will be sent, then the computational capacity of the system will be reduced by n times. Increase in the number of copies without the use of other sending sub-tasks system parameters can significantly reduce computational ability of distributed, but will not achieve the required reliability or speed of obtaining the correct results.

Also, the principle of issuing the copies of a single task is important. For example, you cannot issue copies of a single task to the computing nodes of one user. In order to avoid simultaneous shutdown of nodes with all task copies.

2.4 Results verification

Results verification in accordance with the specifics of tasks is an important tool to reduce the number of copies. The correctness of the task calculation can be verified by comparing the results of a calculation from several computational nodes. But this approach requires at least 2 results. In the event that a meaningful result (e.g., simulation modelling results), you can verify the correctness of the result by partial recount. However, note that you will need dedicated computational power to verify the results. In the case of a search problem or combinatorics, there is no such an opportunity.

Solved problems in a distributed computing system can be divided according to the nature of the results obtained into the following classes:

- Simulation with different initial conditions (large meaningful results, variable length);
- Full bust. Search for one / several solutions (extremely small results, possibly 1 bit);
- Multiple computation of the values of a complicated function (the results are small, the length is the same);
- The method of branches and boundaries. Enumeration with clipping (tasks of varying complexity, small and variable length results);

- Search for parts of a large range (the results are extremely small, you need a guarantee of error-free computing).

One solution to this problem is the validation (data integrity check) of not only results, but results in conjunction with input and intermediate data. Then the verification can be carried out at one result.

2.5 BOINC results validation

By default, the server part of BOINC uses the bitwise comparison of multiple results from different computing nodes (host) for validation of the results. This method is that the BOINC project server makes multiple copies of the problem by default and sends it to multiple hosts, and it compares the results bitwise. If N results are identical, then the server thinks that this result is correct. If it turns out that there were no N identical results [5], the server initiates the generation of new copies of the problem and sends it to several hosts.

This method reduces the computational power of the grid system at least N times. Instead, you can use the mechanism to verify the integrity of the results, using only one copy of the task. It is possible to implement using cryptographic hash functions [6].

Hashing is a convolution of the original data in some combination of some fixed-length by some hash function. The hash function must be cryptographic, i.e., the property of this function unilaterality should be carried out: nobody should be able to select appropriate data by the value of the convolution combination [7].

However, only one copy of the problem still should not be sent, as there is a chance that the result generally will not be counted by the user for the specified period. But to solve this problem, we will still need almost always a lot less copy than if we do not use validation with the hash functions.

In addition to verification of the results integrity, there are still challenges of input and intermediate data integrity. In the case of integrated use of hashing, it`s possible to achieve full data integrity control on the BOINC client side. Let`s look at the different types of attacks on the BOINC client side.

3 Scenarios of attacks on an unprotected scheme

3.1 List of attacks

Using only one copy of problem by all data, including input, output and intermediate ones, it is necessary to guarantee the integrity and secrecy. This is necessary because there are many scenarios of attacks on unprotected data:

1. When a user gets files from BOINC, the user accidentally or intentionally modifies the input file so that the app will still start modelling and will give an incorrect result.

2. In the process of modelling the application after creating one or more checkpoint files, crashes. Then, the user accidentally or intentionally changes the checkpoint file so that the app will still start modelling and will give an incorrect result. Or it just will substitute it with another checkpoint file from another problem.
3. The user may also change a ready a file with the result.
4. The user may replace the file of the finished result by any file at all.
5. The user may accidentally or intentionally modify the configuration file so that the modelling will begin and be completed, but the result will be incorrect.

3.2 Hash functions

Hash functions were used to protect against attacks on the data integrity. A hash sum of the input and configuration files are calculated on the server-side and sent to the host. On the host side during modelling, the hash sum of the checkpoint file and the output file are additionally calculated and together with the rest of the hash sums and the finished result are sent to the server.

To hide the data, two types of encryption are used: streaming and asymmetric. Streaming encryption is used for the input configuration file and the checkpoint file and also for the intermediate output file and the files with hash sums. While asymmetric one is used for the finished output file.

3.3 Standard methods to protect the BOINC platform integrity

The BOINC platform has a built-in function of signing file with an electronic signature on the server before sending them to the user. Private and public keys are used for the signature. The private key remains on the server after the signature and the public one is sent to the user along with the files to verify their signatures. This method can protect only between 1 and 5 attacks.

4 Changed scheme of BOINC project

The scheme of BOINC project functioning was implemented and modified. For the clarity, the scheme was divided into two parts, the scenario on the server side and the scenario on the host side.

4.1 Scenario on the server side

Sending

When a problem is sent to a user, in addition to the input and configuration files, the files storing hash sums are also sent. The input and configuration files are sent in an archive. The archive is encrypted by the stream encryption algorithm before sending.

Receiving

The server receives the ready output file and the archive with the hash sums of the input, configuration and output files from the host. The server decrypts the received data and then calculates the hash sum of the input and configuration files, whose copies are still stored on the server since the problem is submitted to the user and from the ready output file. Then, the hash sums are compared bitwise with the hash sums from the hash sums archive. If at least one of the sums does not match, then the result is considered incorrect.

4.2 Scenario on the host side

After receiving from the server the input and configuration files and an archive with their hash sums, the application is launched.

The application decrypts and extracts the input data and an archive with hash sums, calculates hash sums of the received input and configuration files and compares bitwise them with the hash sums from the hash sums of the hash sums archive. If they match, then the modelling starts.

During the modelling, a checkpoint file is periodically created and an intermediate output file is saved. Before saving these files, their hash sums are calculated and stored in the hash sum archive. Also before saving the checkpoint file and the intermediate output file are archived and encrypted by the stream encryption algorithm.

In addition, there are cases when the files are saved incorrectly, for this, immediately after they are saved, the stored data are read and compared bitwise with those that the app still keeps in its memory.

If an application crashed, the next time it is launched, it will decode and extract checkpoint file, calculate its hash sum and a compare it bitwise with the one stored in the archive with the hash sums. If the sums match, the application will continue modelling since the creation of the checkpoint file, otherwise the modelling will start from the beginning.

After modelling is complete, the ready output file is also archived and encrypted with a symmetric encryption algorithm.

In the end, after modelling completing, the host sends the output file and the archive with the hash sums of the input, output and configuration files to the server.

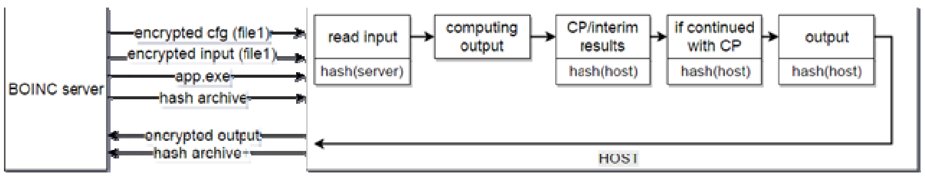

Fig. 1. General scheme of interaction between the server and the host

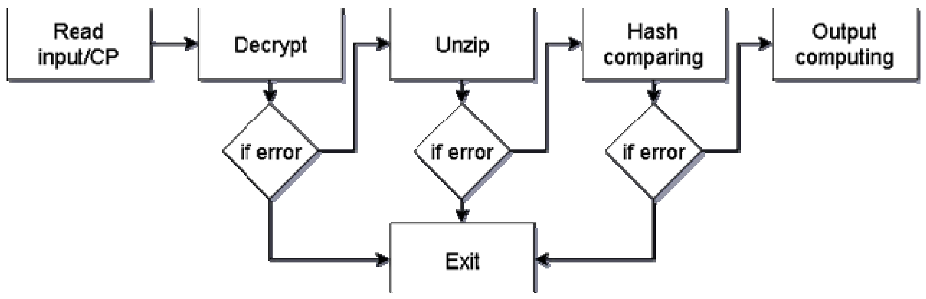

Fig. 2. Schema for reading the input and checkpoint file

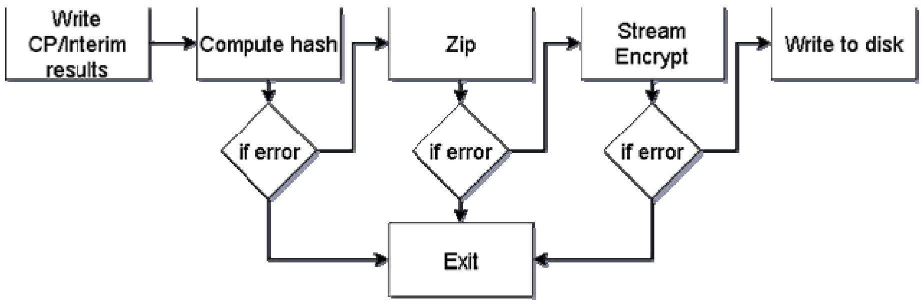

Fig. 3. Scheme for recording the checkpoint file and interim results

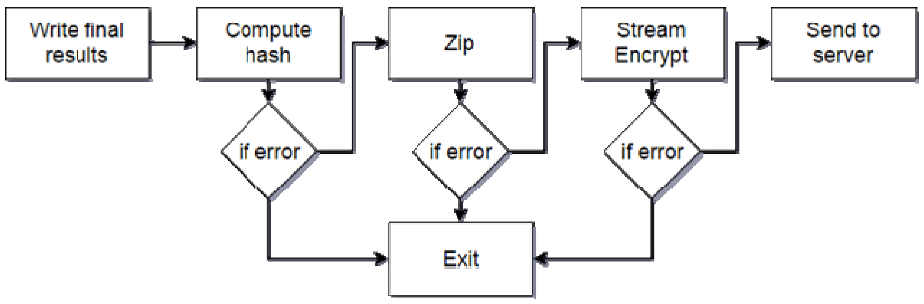

Fig. 4. Scheme of recording the completed output file

4.3 Archiving

The files are archived before encryption every time because if this is not done, then there is some probability that even an encrypted file, after the accidental/intentional changes will remain suitable for use by an application. This can happen if, for example, when decoding only one digit will be changed to another.

The second reason is that archiving before encryption complicates the encryption cracking.

The third reason is that the ready output file can weigh quite a lot for sending it over the Internet, and archiving allows you to reduce the weight about 10 times.

For archiving an open source library SharpZipLib is used (compression level 2).

4.4 Encryption keys

For streaming encryption the sewn source code is used. This is because in the application operation there is a periodic encryption/decryption of the intermediate output file and checkpoint file.

4.5 Application of the proposed methodology

To test this methodology, private BOINC project was developed and as a computational application was used implemented a simulation model of functioning telecommunication network [11]. Different model networks were taken as input data.

5 Conclusion

Using cryptographic hash functions and archiving results allows you to maintain the integrity of the data. Verification of the result is possible by one copy, and the number of initial copies of tasks can be reduced without loss of reliability. As a result, the computing power of the distributed system can be increased.

However, the use of encryption of data and results in voluntary distributed projects can adversely affect their popularity (computational ability). Since for public projects of distributed computing, the policy of openness is mandatory.

References

1. Anderson, D. (2004). BOINC: a system for public-resource computing and storage. *IEEE*.
2. LHC@Home. URL: http://lhcathome.web.cern.ch/
3. Einstein@home. URL: https://einsteinathome.org/ru/home
4. Bertin, R., Hunold, S., Legrand, A., & Touati, C. (2014). Fair scheduling of bag-of-tasks applications using distributed Lagrangian optimization. *Journal of Parallel and Distributed Computing*, *74*(1), 1914-1929.
5. Afanasiev, A. P., Bychkov, I. V., Manzyuk, M. O., Posypkin, M. A., Semenov, A. A., & Zaikin, O. S. (2015). Technology for integrating idle computing cluster resources into volunteer computing projects. In *Proc. of The 5th International Workshop on Computer Science and Engineering, Moscow, Russia* (pp. 109-114).
6. Rogaway, P., & Shrimpton, T. (2004, February). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resis-

tance, and collision resistance. In *International Workshop on Fast Software Encryption* (pp. 371-388). Springer, Berlin, Heidelberg.

7. Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche «The Keccak SHA-3 submission», 2011
8. Posypkin, Mikhail and Semenov, Alexander and Zaikin, Oleg (2012) Using BOINC desktop grid to solve large scale SAT problems. *Computer Science*, 13 (1). pp. 25-34.
9. Vatutin, E. I., Valyaev, S. Y., & Titov, V. S. (2015). Comparison of Sequential Methods for Getting Separations of Parallel Logic Control Algorithms Using Volunteer Computing. *BOINC: FAST*.
10. Vatutin, E. I., Zaikin, O. S., Zhuravlev, A. D., Manzyuk, M. O., Kochemazov, S. E., & Titov, V. S. (2016). Using grid systems for enumerating combinatorial objects on example of diagonal Latin squares. In *Distributed computing and grid-technologies in science and education (GRID'16): book of abstracts of the 7 th international conference. Dubna: JINR* (pp. 114-115).
11. Kurochkin I., Grinberg Ya., Different Criteria of Dynamic Routing, *Procedia Computer Science*, Volume 66, 2015, Pages 166-173.