4. Data Partitioning and Task Management in the Clustered Server Layer of the Volunteer-based Computation System

Jarosław Kuchta

Gdansk University of Technology,
Faculty of Electronics, Telecommunication and Informatics,
Computer System Architecture Department
e-mail: j.kuchta@eti.pg.gda.pl

Abstract

While the typical volunteer-based distributed computing system [1] focus on the computing performance, the Comcute system [3] was designed especially to keep alive in the emergency situations. This means that designers had to take into account not only performance, but the safety of calculations as well. Quadruple-layered architecture was proposed to separate the untrusted components from the core of the system. The main layer (W) consists of independent server nodes, which are coupled into a cluster. The W-servers provide task promotion among the nodes, data partitioning, results gathering, comparing and merging. The cluster remains operational as long as one of the nodes is able to operate.

This paper describes the functionality of the Comcute system from the W-node perspective considering two task parameters: required performance level and required level of computational reliability.

Keywords*: Comcute, volunteer-based computing, distributed computation, data partitioning, task management*

4.1. The Comcute system architecture

The multilayered architecture of a computer system usually is modeled "vertically" with an application layer at top, a computational (also called *business*) layer in the middle, and a data storage layer at bottom [4]. However when we consider the location of the multilayer *distributed* system in the Internet environment, it may be better to model it as a ring (Fig. 4.1). Since Internet is well known for the system designers as an untrusted environment [5], the system core boundaries must be precisely drawn. In the Comcute system architecture [6] Internet users' computers are used to perform calculations (not to store the data). Data is stored in the internal (*core*) layer consisting of multiple *W*-nodes and then transferred to the computers of Internet users (*I*) for calculations. At this point of view the Internet users are *not the clients* for the

Comcute system. Their computers are the critical components of the system and they must be separated from the core *W*-layer by additional proxy layer consisting of *S*-servers. The client for the Comcute system is a *customer* user who orders computations. This user is the most important element of the Comcute system in the sense that the whole system serves him. But the customer is also an *untrusted* element, because he can provide incorrect code to the system and can destroy it. So the customer must be also separated from the core *W*-layer. This is done by the very internal *Z*-layer, which serves as a user interface and validates customer's requests.

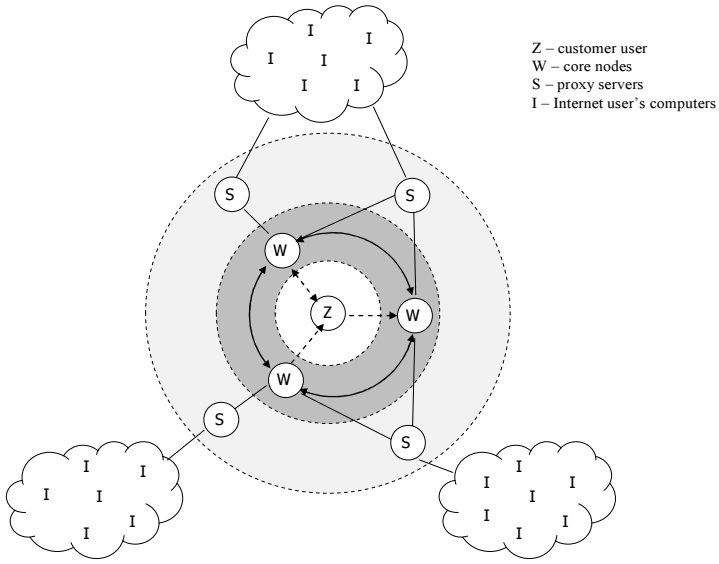

Fig. 4.1. Quadruple-layered Comcute architecture

The computers of the Internet users (*I*) are the most untrusted elements of the system. Users can interrupt their calculations simply by closing the web browser, thus leaving a job unfinished. But even if we assume the volunteers are willing and responsible enough to complete their tasks, their computers may be infected and thus may open the path to attack the whole system for some hostile organization.

To separate the computers of Internet users from the system core, the proxy layer (*S*) is used. Proxy servers not only distribute the computation tasks from the core nodes (*W*) to the Internet computers, but they also control the execution of tasks by the individual computers. In the case of suspicious behavior of a computer, the proxy server can disclose it from the computing group. One group of computers can be supported by a number of proxy servers ensuring network load balancing, thus protecting (to some extent) the core of the system against DDoS attacks.

The main core layer is composed of multiple *W*-nodes distributed in the Internet. The *W*-nodes are organized in a cluster. When a request from a client comes to the cluster, the *W*-nodes are divided into groups and they propagate

the data among themselves. Each group process a part of the dataset. A "group" can be one node, but when a group consists of multiple nodes, they may compare the results of the calculations among themselves to protect the system against the false results provided by the computers of Internet users. The more groups of *W*-nodes are involved in the task, the higher degree of parallelism can be achieved.

Customer user can be a government officer, company manager etc. requesting the computation. This user may communicate with any of *W*-nodes when sending the request. Because the customer is also untrusted (in the sense that the customer can provide an incorrect code), so the user interface is a separated *Z*-layer. The customer user is authorized to execute only approved code.

There is no "central *Z*-node" of the system. *Z*-layer is a virtual server provided by *W*-cluster. Thus as long as one *W*-node can operate, *Z*-layer is accessible for the customer user.

Results of the calculations are stored in each of the involved *W*-nodes. After completion of the calculation the *W*-cluster sends a message to the customer, and the customer may retrieve results from any of accessible *W*-nodes.

4.2. Design requirements

The main design requirements for the Comcute system were as follows [7]:

1. The Comcute system should process huge amounts of data using a scale effect of the very large number of computers of Internet users.
2. The processed data may be confidential.
3. The data will be processed in an *volunteer-computing* mode.
4. The Comcute system can not have a single point vulnerable to attacks.
5. The Comcute system must be resistant to attacks from outside, both on the capture of data, DoS attacks, as well as the deliberate distortion of the results.
6. The Comcute system must provide the ability to continue the calculation, even if most of the nodes will be attacked and eliminated, and in extreme cases up to the moment when a single node is able to act.
7. A customer (user requesting the calculations on the system) can place order to any node and retrieve the results from any other operating node.
8. The calculations can be time consuming - time between order and receiving the results of calculations can be delayed and the moment to obtain the results depends on the client.
9. The system should provide verification of calculations by comparing the results collected from various Internet users. The approved exceptions are the calculations which are non-deterministic.

10. There may be situations possible when the computational task is considered completed even if not all results have been obtained in 100%.

These requirements are the base of the core layer design.

4.3. The structure and role of the core layer of the system

The core layer is composed of independent and cooperating *W*-nodes driven by the same algorithm. Each *W*-node has the following functionality [8]:

1. Taking orders from customers.
2. Distributing of accepted tasks to other *W*-nodes.
3. Task data partitioning into pieces according to the customer's request.
4. Offering the code of the calculation tasks and pieces of data on demand for proxy *S*-servers, which subsequently distribute them to the computers of Internet users.
5. Gathering the results of calculations reported by *S*-servers from the Internet computers.
6. Synchronizing calculations with other the *W*-nodes and comparing the partial results gathered from other nodes in accordance with the arbitration algorithm specified by the client.
7. Ability to reconfigure the system against attacks from the outside.
8. Informing the client about the state of performing the task on demand.
9. Determining of the task completion in accordance with the user-defined stop condition.
10. Merging the results of calculations according to the algorithm specified by the client.
11. Signalizing to a client about the task completion.
12. Storing and providing the results rendering accessible data available for the client.

4.3.1. Task management by customers

A customer may submit an order to any *W*-node within the Comcute system. The ordered computational task must be chosen from a predefined (by a trusted admin user) set of task classes. Task class defines:

1. program code for data partitioning,
2. program code to calculate the data subsequently distributed to the computers of Internet users,
3. optional program code for comparing the results among *W*-nodes and resolve their non-compliance (arbitrator),

4. optional program code determining the end of the calculation (stop-condition),
5. optional program code to merge the partial results obtained from the computers of Internet users into the final result set (linker),

The same computational task order should include the required level of performance (*QoS – Quality of Service*) and the required level of reliability of the results. The QoS parameter indirectly determines the number of *W*-nodes involved in the task. The customer can not directly determine this number, because the Comcute system must provide resistance to the attacks and has to be able to work event when only one *W*-node is operational. This means that the system must be able to ensure its operation event at the expense of the required level of QoS.

The required level of reliability indirectly determines the number of calculations to be carried out independently with the same piece of data (i.e. the number of *W*-nodes to process the same piece). Three levels of reliability are offered:

1. Any piece of data should be calculated only once.
2. Two parallel calculations are started at one piece of data. In the case of non-compliance results, the calculations are restarted.
3. Three parallel calculations are started at one piece of data. When two *W*-nodes get the same result, this result is sent to the third node. In the case of non-compliance results among the two nodes, the third calculation will be conclusive.

The higher level of required reliability can be used only in the case of deterministic calculations. If the calculations are non-deterministic (e.g. weather forecast) and the higher level of reliability is required, the customer can provide an advanced arbitrator code used to verification of the individual results (for example using fuzzy logic). If the nodes can not agree on the results at a higher level of the required reliability, the number of calculation attempts may be specified. The third level differs from the second one in that sense that it protects the system against an of an “false computing” attack (the attacker must provide more than 66% false results to perform the attack successfully).

The required level of reliability higher than the first increases the cost of providing QoS twice and three times.

4.3.2. Distributing the tasks among nodes

According to the QoS parameter and the required reliability level the *W*-node, which was given the task (W_0), must propagate it to the appropriate number of other nodes. Thus it needs to verify the status of all other nodes. Each node can participate in multiple tasks, engaging multiple proxy servers and an undetermined number of Internet computers, so the node status must contain not only the number of tasks, but also the real load of the node and the real efficiency of the task performance.

The W_0-node calculates the number of nodes needed to ensure both the required QoS parameter and the required reliability level as:

$$N_K = N_Q \cdot N_W$$

where:

- N_Q is the number of nodes required for QoS level and for number of data packages predicted by the customer (the customer should provide an approximate number of packages when he places an order),
- N_W is the number of nodes required to ensure a level of reliability.

If N_K is less than or equal to the total number of available nodes (N_C), the subset W' of the available nodes will be formed. The W'-set contains the number of N_K least loaded nodes. Otherwise, N_Q is reduced until $N_K \leq N_C$. In the extreme case where $N_Q = 1$, N_W is reduced to 2 or 1 (if it is possible) thus reducing the level of reliability. If only N_K is greater than 1, the subset W'' contains more than one node and W_0-node must propagate the request to all other nodes of the W' set.

If N_W is greater than 1, then the W''-set is divided to N_Q groups, calculating the same pieces of data and verifying the results with each other. If $N_W = 1$, then these groups are of one-node and cross verification does not occur. This is a simplified situation, and some activities of the algorithm (shown below) can be omitted.

4.3.3. Data partitioning

Partitioning of the data received from the customer to pieces adjusted for the parallel calculation is determined by the class of the task and the code of the partitioner. It depends only on the logic of the task and it does not depend:

- neither on the number of groups of nodes resulting from the task promotion (N_Q),
- nor on the number of nodes in each group (N_W).

If the number of N_Q groups of nodes involved in the task is less than the number of data packages, then the whole data set is divided to *subspaces* and each group processes one subspace. If the number of N_Q groups is greater than the number of data packages, then outnumbered groups are ommited.

All nodes partition the data by the same algorithm, which further allows to compare the partial results of calculations returned by Internet computers. The data is stored in a data repository as a vector of data packets at each node independently.

Along with the data partitioning each node creates its calculation state vector, which will include the status of calculations for each packet of data. This vector consists of the following states:

- package not sent (initial state),
- packet sent,

- result received,
- correct result,
- incorrect result.

A simplified diagram of state transitions for the data processing in each node is shown in Fig. 4.2.

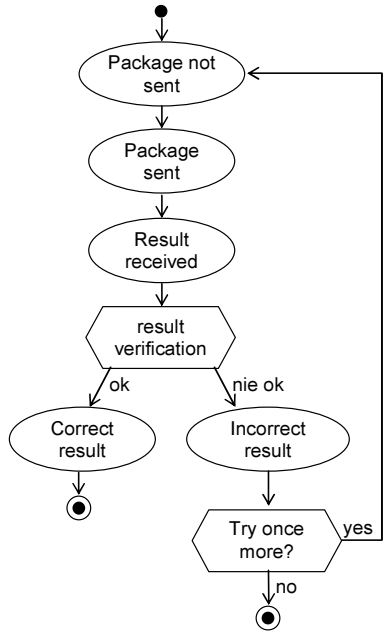

Fig. 4.2. State transition diagram for the data processing

4.3.4. Distributing the computational code and data packages to proxy servers

The code of the calculation defined by the customer must be delivered by *W*-nodes to the computers of Internet users on demand from *S*-servers. This code is the same throughout the task (even for a whole class of tasks), so it can be delivered to any Internet computer only once, prior to the first package of data.

Each of the *W*-nodes provides a package of data from its data repository to the proxy *S*-servers. Each N_Q group provides data packets from its data subspace. If all the data packets is marked as N_D, the number of packages in the subspace is equal to:

$$N_P = \text{Int}(N_D/N_Q)$$

If $N_Q \cdot N_P < N_D$, then one of the data packages is larger than others, and a group of *W*-nodes it supports will be more loaded than other groups (which will have to wait a little longer for this very group to complete the calculations).

Group ID assigned to the *W*-node determines a data subspace from which it provides the data package. The calculation of the exact location of packet in the subspace may require additional algorithm.

If the required level of reliability equals to 1, a group of nodes are of one-node and single node forming a "group" provides a package of data independently of other groups – nodes. One can use the pseudo-random selection algorithm within the subspace of data packets, which reduces susceptibility to repetitive attacks from the outside (Fig. 4.3a).

If the required level of reliability is greater than one, each group contains two or three nodes. Within each group, individual nodes provide data packets in the same order. This can be achieved by initializing the pseudo-random selection algorithm with the same initial value (*seed)* within the group of nodes (Fig. 4.3b).

For the selected packets the state of calculation is examined. Those packets for which the state is equal to "package not sent" or equal to "incorrect result" and a flag "try once more" is true are sent to the computers of Internet users.

When the data is sent the node changes the data package its status to "package sent". If the previous calculation state was "incorrect result", the attempt counter is decremented an when it reaches zero the flag "try once more" is set to false.

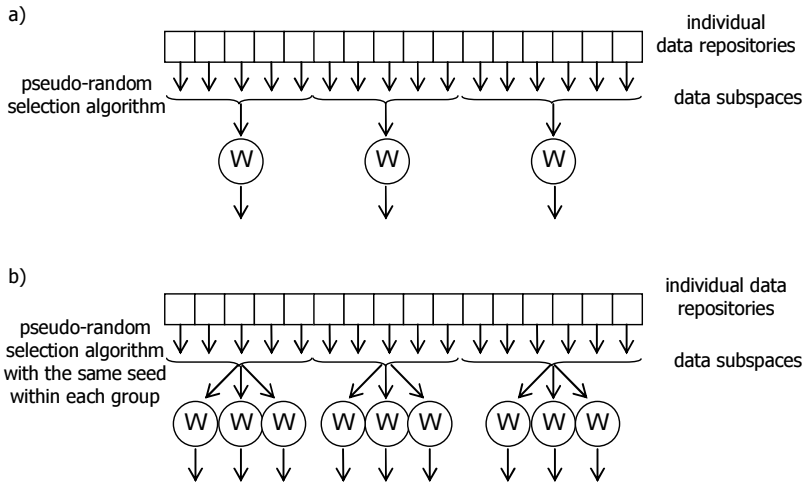

Fig. 4.3. Data packages selection for: a) one-node groups, b) multi-node groups

4.3.5. Gathering the results of calculations from the proxy servers

The results of calculations for each data package are returned to its origin *W*-node. The *W*-node then needs to store the result in its own repository (in a form

of vector of partial results) and to change the status of the packet to “result received”.

If the required level of reliability is equal to 1, then:

- When the arbitrator code was specified (in this case the arbitrator code must simply verify code results with the data), this code is executed. If the verification code confirms the reliability of a result, the package status changes to “correct result”. If not, then it changes to “incorrect result”.
- When the arbitrator code was not specified, each result changes the package status to “correct result”.

If the required level of reliability is higher than 1, then the multiple nodes synchronize calculations and compare results within a calculation group.

4.3.6. Synchronizing calculations with other the *W*-nodes and comparing the results

All nodes involved in the task performance (of the W’-set) send to other nodes queries for the vector of calculation state from time to time. This vector is used to determine which results (for which packet) should be sent among the nodes.

Results can be loaded by the node for all data packets except those which state in their own state vector is equal to “correct result”.

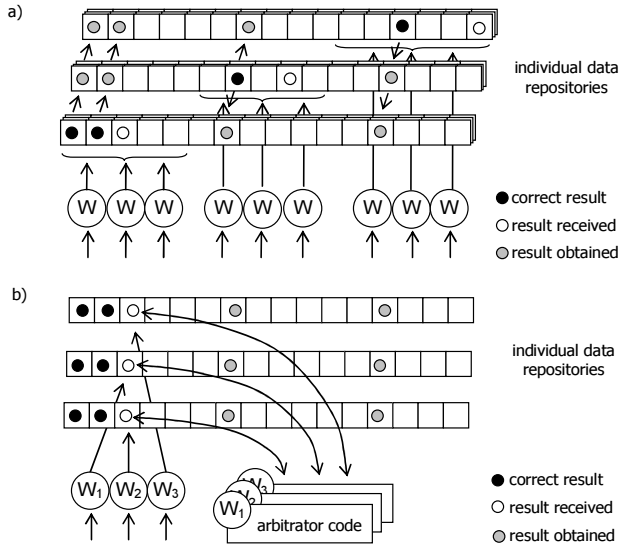

Fig. 4.4. Synchronizing calculations among nodes: a) collecting correct results from other groups of nodes, b) comparing results among the nodes within each group

First those results which state at another node is equal to “correct result” are collected from those nodes, These results are collected from all nodes belonging

to the W'-set (Fig. 4.4a). For these results, an additional flag "result obtained" is set to true.

Second, those results which status is "result received" in all the vectors of the nodes in the current group are collected from the nodes belonging to the same group (as long as the required level of reliability is greater than 1). These results are compared using the arbitrator code (Fig. 4.4b). If the arbitrator determines that the node self result is valid, then it changes the status to "correct result". If it determines that is invalid but the result of another node is valid, then the own result is rewritten to the valid one. The status is also changed to "correct result". If an arbitrator is unable to determine the correct result, the state of the calculation will change to "incorrect result".

4.3.7. System reconfiguration to empower resistance against attacks

In the absence of threats from outside the results are verified within the group. All nodes within the group supports data packets belonging to one subspace data. Other groups support their data subspaces in parallel. As far as results are received and verified, they are propagated from each group to other groups. However, in this scenario, there may be two disruptions resulted either from external attacks, or from internal failures within the system:

1. Node may not be able to verify the results within the group.
2. One or more groups of nodes may stop working and will not transfer their results to other groups.

The first disturbance may be manifested in two ways:

1. One or two nodes of the group (for the required reliability level of 2 or 3) stop sending results and the results may not be compared.
2. The results of calculations are non-deterministic and the system can not determine the correct result.

In the first case the node, which itself gathers the results from its Internet computers, but not obtains the results from other nodes in the group, tries to co-opt other nodes (both belonging to the W'-set and outside of it) to their own group. This node collects load status of other nodes and sends them an invitation to the compute group. It may happen, however, that none of the invited nodes will not be able to join the group. Since the system can not provide the required level of reliability, the number of groups must be reduced. The results already calculated and verified as correct shall be disseminated to all groups and are stored in repositories. This operation is performed until two or three nodes stay operating, and these nodes may form one group large enough to assure the required level of reliability. If the remaining number of nodes is not efficient enough to provide the required level of reliability, the system control at each node may decide:

- to continue the calculation at a lower level of reliability,
- to suspend the calculation until the system will be able to provide the required level of reliability,

- to send the previously calculated results to the customer with a request to change the task parameters.

In the second case (which can occur for any required level of reliability) the calculations will be automatically continued if the security level is equal to 1 and the customer has not specified the special arbitrator. This situation is consistent with the requirements of the client. In another situation, the system control must decide as above.

If any of the groups of nodes ceases to deliver results, then the system control should create a new data space of the other packages the data, reconfigure the *W'*-set on the remaining data and resume the calculations.

4.3.8. Informing the client about the task state

Because the task can take hours or even days, so each node involved in its execution (belonging to the *W'*-set) provides the status of the job for the client. If the client request goes to a node that does not belong to the *W'*-set, then that node must propagate a query to other nodes.

On the basis of the state information the customer may decide:

- to continue the calculation,
- to stop the calculation,
- to collect the partial results of current calculations.

4.3.9. Determining of the task completion

By default, the task is completed when 100% of the results will be calculated. The customer may, however, specify some special termination condition depending on the problem. For example:

- When breaking a password - when one of the nodes reports that the correct password is found.
- When searching sites in terms of specified keywords – when the specified number of pages is addressed.

Additionally, the customer may specify the timeout for the results. After this time the task must be considered completed and the system starts merging the results.

4.3.10. Results merging

By default, the partial results returned by the computers of Internet users for each data packet (and stored in the repository) are considered to be the final results of the task. In this case, the final result is a simple list of the partial results corresponding to recently established calculation state vector.

The customer may, however, specify an additional algorithm for merging the results. This algorithm runs on a data repository and produces the needed final

result. However, this algorithm must be simple enough to be executed efficiently on each *W'*-node.

In some applications, the client may request that the results of calculation have to be used as input for another task. Then the merge algorithm can:

- sequentially run a different, explicitly given task,
- recursively run the same task again, but with the new data resulting from the previous task.

4.3.11. Informing the client about the task completion

The customer may independently check the status of the job. But if he provide his e-mail address, then one of the *W'*-nodes send the notification of the completion of tasks to this address and at the same time notify the other nodes of this fact (to avoid the avalanche of notifications sent to the client).

4.3.12. Storing and providing the results of calculation

The calculation results are not sent by mail to the customer, but are stored in the *W*-nodes (in the individual repositories) at least until the client downloads them to his own computer. For safety, the results should be stored at all nodes (not just at these nodes which belong to the *W'*-set).

The client may choose to have long-term storage of results, although this should be adequately paid because of the high consumption of non-volatile memory belonging to the system.

Bibliography

1. Nov O., Anderson D., Arazy O.: *Volunteer Computing: A Model of the Factors Determining Contribution to Community-based Scientific Research.* WWW 2010. Raleigh, USA
2. BOINC. home page http://boinc.berkeley.edu
3. Balicki J., Paluszak J., Zacniewski A.: Selected architecture aspects of the distributed computer systems. pp.7-23 in Distributed computations in grid architecture systems, Balicki J., Kuchta J. eds., 2012, in Polish
4. Dennis A., Wixom B.H., Tegarden D., Systems Analysis & Design. An Object-Oriented Approach with UML, John Wiley and Sons, USA, 2002
5. Brown A., Ryan M.: Synthesising Monitors from High-Level Policies for the Safe Execution of Untrusted Software, pp 233-247 in Information Security Practice and Experience, Chen L., Mu, Y., Susilo W. eds, LNCS, 2008
6. Matuszek M.: The Comcute system architecture. pp. 83-92 in Distributed computations in grid architecture systems, J. Balicki, J. Kuchta eds., 2012, in Polish.

7. Balicki J., Matuszek M., Szpryngier P., Kuchta J., Czarnul P., Szymański J., Brudło P.: The Comcute system functional design, tech. rep. WETI PG, 33/2011 in Polish
8. Kuchta J.: Propagation and Synchronization of Computations among Nodes. pp.101-113 in Distributed computations in grid architecture systems, Balicki J., Kuchta J. eds., 2012, in Polish