American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-11, Issue-02, pp-06-11
www.ajer.org
Research Paper Open Access

Design of hardware and software complex of fault-tolerant computing by iterative methods

Denis Zolotariov
Ph.D. in Physics and Mathematics, Kharkov, Ukraine, ORCID: 0000-0003-4907-7810,

ABSTRACT : *The approach to creation of a software and hardware complex for fault-tolerant calculations using iterative numerical methods for solving problems of various natures based on a flexible multi-node infrastructure and software orchestration of calculation nodes is presented. In addition to protecting against errors during execution of a computation algorithm, application of the developed approach also allows manually and automatically stopping and restarting this process. The proposed approach has self-healing for cases of corrupted, missed or duplicated saved results.*

KEYWORDS: *fault-tolerant computing, high-availability, numerical methods.*

Date of Submission: 26-01-2022 Date of acceptance: 07-02-2022

I. INTRODUCTION

A feature of the algorithms of iterative analytical, numerical and analytical-numerical methods is the natural ability to preserve their state during operation. Using this opportunity in software design – saving after the end of each iteration its results, – it is possible to load and process the results of all previously calculated iterations, thereby completely restoring the state of the program for the last calculated iteration of the algorithm. Such saved results will be the program's high-availability (HA) journal, which allows restarting the algorithm from the last correctly completed iteration in case of a computation failure, instead of restarting it entirely.

Moreover, for this purpose, the iterative nature of the algorithm is not important. It can be either an initially iterative method (for example, Newton's method [1]), or a method reduced to an iterative form. An example of the latter is the approach described in [2], for the approximating functions method [3,4], which was brought to an iterative form by dividing the domain of the problem definition into "windows". In other words, non-iterative algorithms that can store and load the results of intermediate computations as their state can also be considered iterative.

But HA journal alone is not sufficient to create such fault-tolerant software. For a comprehensive solution to the problem of fault-tolerant calculations by iterative methods, which is the subject of this article, it is necessary to solve the following three problems: formation of a HA journal, preparation of a set of computational nodes, as well as software responsible for the automatic management of these nodes. This article is devoted to solving these problems, using system analysis and modern tools for creating and managing virtual servers.

II. PROBLEM STATEMENT

The solution to the above tasks can be represented in the form of the following functional elements presented on Fig.1.

The first element is a HA journal generated from the results of each iteration of the numerical method algorithm or parts of an iteration – steps. Its mechanism must assume and correctly handle corrupted, missing and duplicated data–“Filter” on Fig.1. It is responsible for transferring state from and to worker nodes.

The second one is a set of prepared nodes for a computational cluster that performs calculations by the method in the form of active worker nodes and standby ones.

The third element is an orchestrator over this set of computational nodes, which is responsible for identifying and decommissioning damaged nodes when an accident occurs on them, and launching new ones from the reserve to replace them.

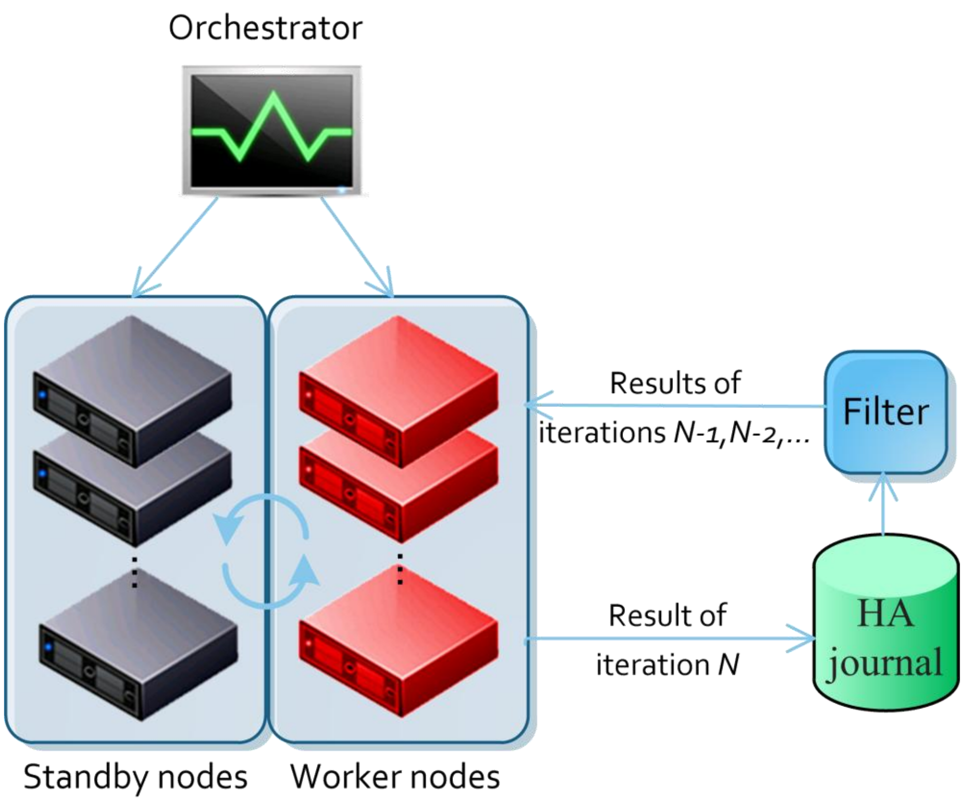

Fig. 1. Elements of computational system

Together they represent the proposed hardware and software complex for creating a fault-tolerant system for solving problems by iterative methods. Let's define in more detail the requirements for each element of the system.

III. JOURNALING FOR HIGH-AVAILABILITY

To maintain a correct HA journal, the algorithm of the method must not just be broken down into iterations, each of which dependsonly on one or a few previous ones, but each iteration, if possible, should be broken down into steps with connection to only one or a few previous ones. In the simplest case, iterationhasonly one step inside.A diagram of such a partition of the algorithmis shown in detail in Fig. 2, where*i* is a currently executing iteration of the algorithm, *s* is the currently executing step of this iteration, and:

$$1 \le m \square \; i , \tag{1}$$

$$1 \le l \square \; s . \tag{2}$$

The results of the execution of iterations and their steps must be stored in a fault-tolerant storage, which at the program level should be a database (for example, SQL-databasePostgreSQLor noSQL oneRedis) or, in some cases, a queue with the possibility of sequentially fetching the required set of iteration and step results(for example, Apache Kafka). It is recommended to choose the most fault-tolerant enterprise solutions operating in real time.

Moreover, the repository should be the only one, following the paradigm of a single source of truth, in the role of which it acts, despite the fact that this approach leads to a longer restoration of the state by the program when the interrupted computation continues.

At the physical (bare metal) level, such storage must form a fault-tolerant distributed cluster of several independent servers that minimizes the probability of their failure at the same time. According to modern recommendations [5], for the possibility of uninterrupted operation in case of failure on *N* servers from the cluster, the cluster must include at least k_{nodes} servers:

$$k_{nodes}^{HA\ journal} = 2N + 1 . \tag{3}$$

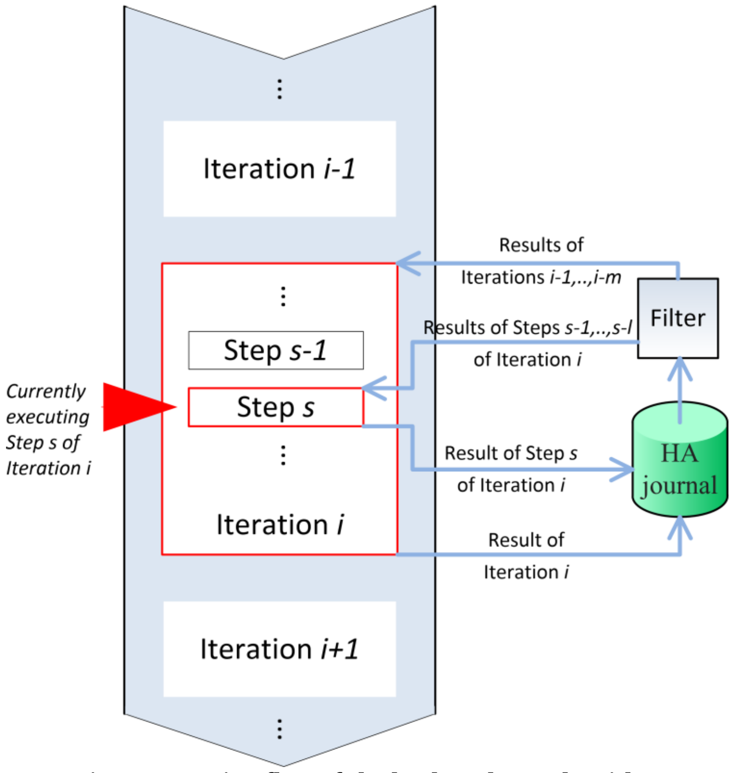
Iteration *i-1*
Step *s-1*
Step *s*
Iteration *i*
Iteration *i+1*
Currently executing Step s of Iteration i
Results of Iterations *i-1,..,i-m*
Filter
Results of Steps *s-1,..,s-l* of Iteration *i*
Result of Step *s* of Iteration *i*
HA journal
Result of Iteration *i*

Fig. 2.Execution flow of the broken down algorithm

Ultimately, this approach to the HA journal should allow, upon failure in any iteration, to restart only that iteration without the need to execute the previous iterations. In case of failure at a certain step – similarly, restart the execution of calculations, starting from this step of the last iteration.

The storage should also be responsible for automatically checking the completeness of the stored data, as well as for data deduplication that can appear in the event of network failures ("Filter" on Fig. 2). With frequent duplication of data, two storages should be created: the first one – temporary – only for writing the state, the second one – main – for reading only, as it present on Fig. 3. The intermediate service reads data from the first one, deduplicates it and transfers it to the main one. In the case of using Apache Kafka, such a service can be Kafka Stream.

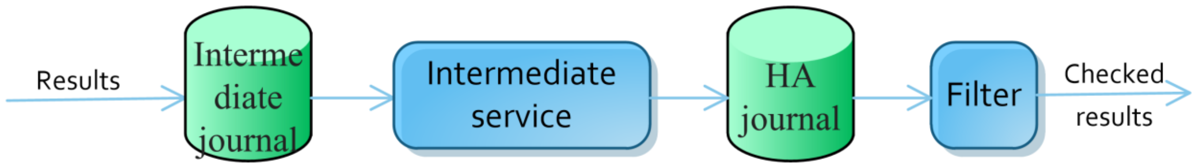

Fig. 3.Results deduplication scheme

Using this approach to building a HA journal, the following advantages appear. Calculations in case of failure or restart can be restored and resumed automatically starting from last completed iteration or iterationstep, since the data is stored in a form ready for loading by the program as initial data, including the continuation of calculations on other servers. When resuming, the program should not load the results of all iterations, but only the last ones, which are actually needed to continue the interrupted calculation.

In addition, the saved data can be used to track the progress of computations in real time when using third-party clients built, for example, on microservice architecture. And, as a result, a HA journal configured in this way allows easy duplication of the tracking microservice in the case of a failure on the working copy, and guarantees the preservation of the iteration results data even in the event of a physical failure in any of its components.

IV. COMPUTATIONAL NODES

A computational node is a server, virtual or physical, that executes computational program and stores its results in an external storage – the HA journal. For ease of management, the nodes should be virtual, but the physical base (bare metal) for them should be computers, built on the basis of modern requirements for fault-tolerant enterprise servers: enterprise data storage systems based on RAID, RAM with ECC and others.

Node images may be the most common Docker images today, or more complex, but slower to deploy, virtual machine images or VPS snapshots can be used. For simple tasks, it is highly recommended to use Docker images.

In addition to running nodes, to reduce downtime, there should also be reserve nodes. Node reservation must be at least one of three types, presented on Fig. 4.

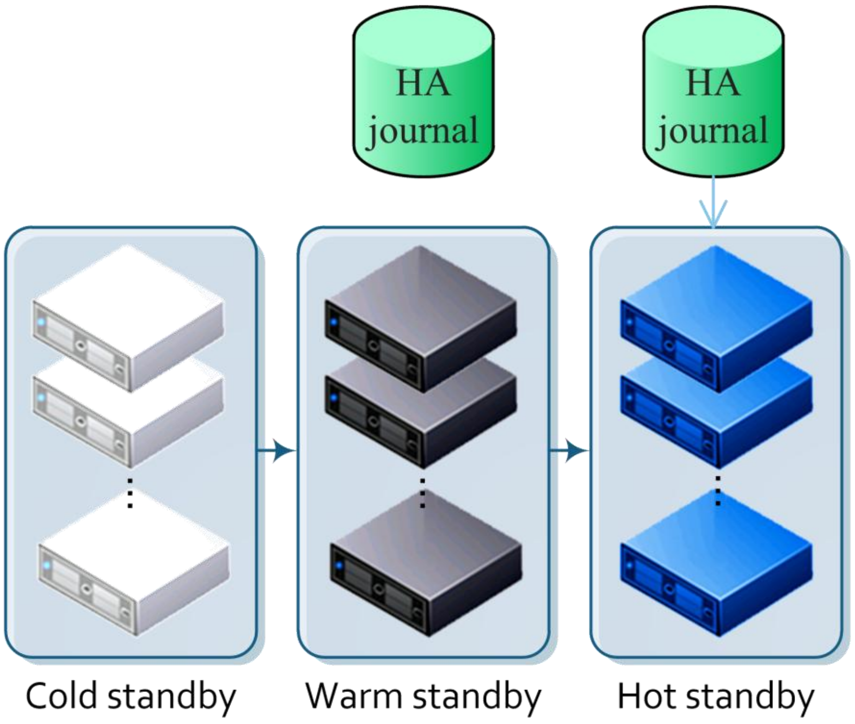

Fig. 4.Standby nodes

Cold standby: a node image containing customized software for performing computations. Expands to a new stateless node on demand and loads the saved state from HA journal and continues calculations at the command of the orchestrator.

Warm standby: a stateless deployed cold standby node. Loads state on demand for inclusion in computations at the command from the orchestrator.

Hot standby: a warm standby node that continuously updates its state from the HA journal as the calculation progresses on the worker nodes. It differs from worker nodes only in that they do not continue the calculation. It can be put into operation most quickly.

Combined standby: some nodes are in hot standby, some are in warm standby. The minimum quantity of nodes is 3: worker node, hot standby node, image in cold standby. If a working node fails, a hot standby node takes its place, and the node itself becomes a warm standby after restarting. The previous warm standby node becomes the hot standby one.

In any case, the most optimal ratio between working and standby nodes is as follows:

$$k_{nodes} = k_{nodes}^{worker} + k_{nodes}^{hot\ standby} + k_{nodes}^{cold\ standby} \geq 3\,, \tag{4}$$

$$k_{nodes}^{worker},\, k_{nodes}^{hot\ standby},\, k_{nodes}^{cold\ standby} \geq 1\,. \tag{5}$$

So, the minimum number of nodes is three: worker, hot standby and cold standby.

It is worth noting that when the program algorithm is executed in parallel on several nodes, the state is saved still in a common HA journal. The result of each iteration and step is marked in such a way that the order in which they are written to the storage or when they are duplicated does not matter. Also, to fully implement the parallelism of the algorithm execution, it is necessary, as usual, to create an external task scheduler for parallel nodes.

The centralized filter software can be supplemented or completely replaced by a per node software, which is responsible for additional validation of the loaded iteration and step results from HA journal.

In the case of missing or corrupted results, the program should take only the results up to the first complete, inclusive, and discard the rest. Its execution will start as if the discarded iterations and steps have not yet been calculated.

V. NODE ORCHESTRATION

We recommend using the time-tested Kubernetes-based generic approach to manage nodes as Docker images.

More sophisticated VPS and cloud-based solutions should use an image management API, an orchestrator based on, for example, Apache Zookeeper [6] and custom software solutions. For example, for the DigitalOcean provider, these will be running Droplets as worker nodes (as described, for example in [7,8], Droplet snapshot as cold standby node, DigitalOcean API as a tool for orchestrator for managing Droplets and Droplet images. The general approach to the functioning of a node orchestrator is to follow the rules below and described in the form of a diagram in Fig. 5.

The orchestrator must poll the nodes with a certain frequency to determine their "alive" status. If a node does not respond positively within a timeout determined by the network and node internal software delays, it is marked as inactive and the process of replacing the node with a standby one is started. The algorithm for replacing nodes with standby ones is as follows.

In case of failure, the calculation simply switches to the first hot standby one. If it is absent, the warm standby node is used, and if the latter is absent, the new node is deployed from the cold standby.

After the hot standby node is included in the process, one warm standby node turns on as a hot standby one (continuously pulls up the state). A new warm standby node is created from the failed one after its restarting. If this is not possible, a new warm standby node is deployed.

If the same failure is repeated on the next worker node, the calculation is interrupted, because the error is not in the node, but in the program algorithm, and a notification is sent.

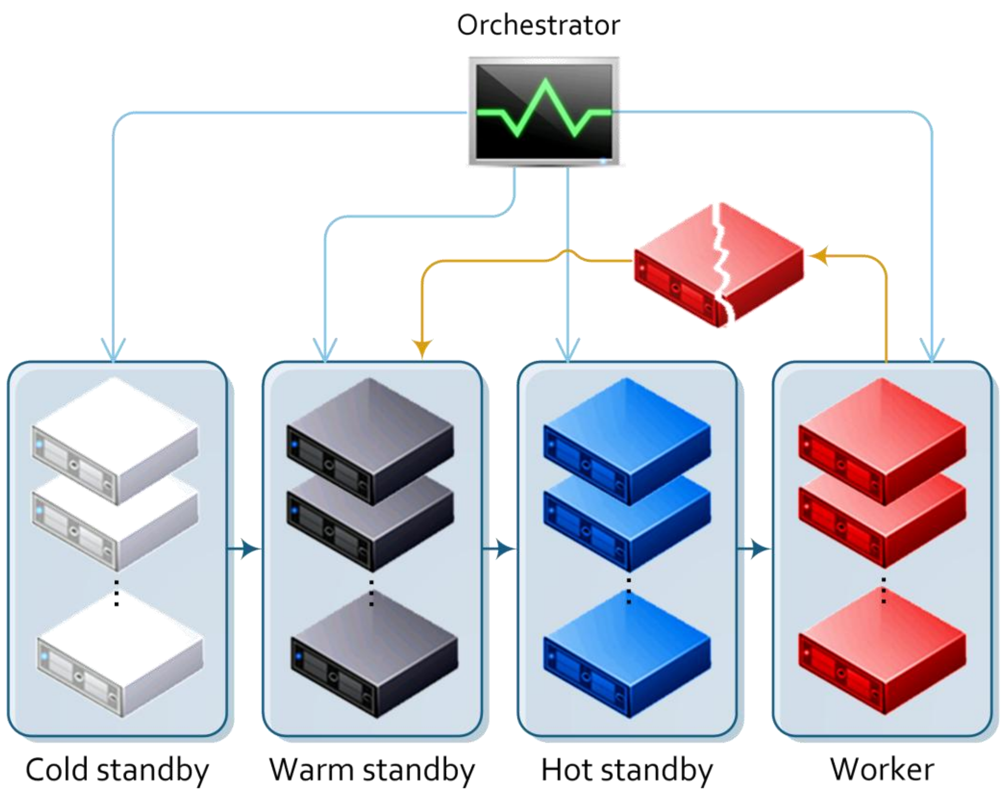

Fig. 5. Nodeorchestratorworkflow

VI. CONCLUSION

The approach to creation of a software and hardware complex for fault-tolerant calculations using iterative numerical methods for solving problems of various natures based on a flexible multi-node infrastructure and software orchestration of calculation nodes is presented. This comprehensive solution is based on solving the following three problems: formation of a high-availability journal, preparation of a set of computational nodes with standby ones of four types, as well as software responsible for the automatic management of them. These problems are discussed in detail.

The proposed approach is suitable for both natively iterative methods and methods reduced to iterative form. The algorithm of the original method is broken down into a sequential chain of iterations with a number of sequential or parallel steps, each of which is built fault-tolerant and stores the results of execution in fault-tolerant high-availability storage. Also, the proposed approach has self-healing for cases of corrupted, missed or duplicated saved results.

REFERENCES

[1] Deuflhard, P.: Newton Methods for Nonlinear Problems. Springer (2011), DOI: https://doi.org/10.1007/978-3-642-23899-4

[2] Zolotariov, D.: The New Modification of the Approximating Functions Method for Cloud Computing. International Journal Of Mathematics And Computer Research 9(9), 2376-2380 (2021). DOI: https://doi.org/10.47191/ijmcr/v9i9.01

[3] Zolotariov, D., Nerukh, A.: Extension of the approximation functions method for 2D nonlinear Volterra integral equations. Applied Radioelectronics 10, 39-44 (2011).

[4] Nerukh, A., Zolotariov, D., Benson, T.: The approximating functions method for nonlinear Volterra integral equations. Opt Quant Electron 47, 2565-2575 (2015), DOI: https://doi.org/10.1007/s11082-015-0141-2

[5] Apache: Apache Kafka (2022), https://kafka.apache.org/documentation/.

[6] Apache: Apache ZooKeeper (2022), https://zookeeper.apache.org/.

[7] Zolotariov, D.: Microservice architecture for building high-availability distributed automated computing system in a cloud infrastructure. Innovative Technologies and Scientific Solutions for Industries 17,13-22 (2021), DOI: https://doi.org/10.30837/ITSSI.2021.17.013

[8] Zolotariov, D.: The platform for creation of event-driven applications based on Wolfram Mathematica and Apache Kafka. Innovative Technologies and Scientific Solutions for Industries 16, 12-18 (2021), DOI: https://doi.org/10.30837/ITSSI.2021.16.012