J. Comp. Int. Sci. (2017) 8(3):153-162
http://epacis.net/jcis/PDF_JCIS/JCIS-0132.pdf
jcis@epacis.net

Pan-american Association of Computational Interdisciplinary Sciences

Quandoop: A Classical Simulator of Quantum Walks on Computer Clusters

David S. Souza [a], Franklin de L. Marquezino [a] and Alexandre A. B. Lima [a1]

[a]Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil

Received 06/11/16 / Accepted 05/08/17

Abstract

We describe software Quandoop, a novel simulator of discrete-time quantum walks using high-performance computing (HPC) with the goal of handling simulations with very large Hilbert spaces. Those simulations are usually restricted to a relatively small number of steps due to memory limitations or reduced processing power. The most important feature of Quandoop when compared to previous simulators is to provide a low cost solution for simulations of quantum walks with extreme memory requirements. Examples of such simulations include quantum walks on fractals or with multiple interacting walkers.

Quandoop uses Apache Hadoop to parallelize the calculations on a computer cluster, thus reducing the amount of time required to perform the quantum walk simulation and allowing its execution with more steps. Our simulator takes the input as text files containing the initial state, the evolution operators and the description of simulation parameters. This approach allows the integration with other quantum walk simulators such as HiPerWalk or QWalk. After running the necessary calculations in parallel, Quandoop outputs text files describing the results associated to the quantum walk.

Keywords: quantum algorithms, quantum walks, simulation, high-performance computing, distributed computing, map/reduce.

1. Introduction

Quantum walks are the quantum counterpart of random walks. The description of a quantum walker, therefore, should be consistent with the postulates of quantum mechanics. Analogously to random walks, which are applied to the development of classical algorithms, quantum walks have also been successfully applied to the design of quantum algorithms [1, 2, 3]. However, the design of efficient quantum algorithms is not the only application of quantum walks. Recently, it has been proved that quantum walks with multiple walkers are universal for quantum computation [4], and hence they

[1]E-mail Corresponding Author: assis@cos.ufrj.br

doi: 10.6062/jcis.2017.08.03.0132

are also a potential technique for the design of quantum hardware. Besides the computational applications, quantum walks are also interesting from the point of view of physics, as a framework to study subjects such as thermodynamics [5] and molecular bidings [6], for instance. In spite of its importance for the development of quantum computing, current technology still imposes strong restrictions on the implementation of quantum hardware. Meanwhile, the only alternative for studying the behavior of quantum walks is to make numerical simulations on classical computers. However, quantum walks may be very hard to simulate classically, since the required memory may increase exponentially with the size of the input.

In this work, we describe Quandoop, a classical simulator of discrete-time quantum walks using high-performance computing (HPC) with the goal of handling simulations with very large Hilbert spaces. Those simulations are usually restricted to a relatively small number of steps due to memory limitations or reduced processing power. Quandoop uses Apache Hadoop, a popular Big Data processing framework, to parallelize the calculations on a computer cluster, thus reducing the amount of time required to perform the quantum walk simulation and allowing its execution with more steps. The most important feature of Quandoop when compared to previous simulators [7, 8] is to provide a low cost solution for simulations of quantum walks with extreme memory requirements.

This work is organized as follows. In Sec. 2, we review the definitions from quantum and distributed computing that are necessary for the understanding of the paper. In Sec. 3, we introduce Quandoop, the quantum walk simulator that is the main contribution of our paper. In Sec. 4, we describe some numerical experiments obtained from our simulator running on a computer cluster. In Sec. 5, we present our closing remarks.

2. Quantum Walks and Map/Reduce

According to the postulates of quantum mechanics, the state of a quantum walker should be represented by a unit vector on a Hilbert space, and its evolution by a unitary operator over that space. The coefficients of the state vector in a certain basis of the Hilbert space are called *amplitudes*. The measurement is represented by a self-adjoint matrix, called *observable*. The outcomes are eigenvalues of the observable, and the state after the measurement is projected onto the corresponding eigenspace.

There are at least three different models of quantum walks. The *discrete-time coined model* has been defined by Aharonov *et al.*, and uses two Hilbert subspaces to keep track of both the coin and the position of the walker [9].

Notice that in the discrete-time quantum walk, the coin can be in a superposition of states. Moreover, the coin in the discrete-time quantum walk is deterministic. In fact, the probabilistic nature of quantum mechanics takes place only when a measurement is performed. The *continuous-time model* has been defined by Farhi and Gutmann [10], and is based on the idea of using the Laplacian matrix of a graph as a Hamiltonian in the Schrödinger equation. Finally, the *discrete-time staggered model* has been recently defined by Portugal et al. [11], and evolves according to reflection operators defined from certain partitions of the vertices of the graph. The staggered model generalizes the previous Szegedy's model [12]. In this work, we concentrate on coined quantum walks, although Quandoop is also capable of simulating staggered quantum walks and other oracular quantum algorithms.

The need for processing large amounts of data is increasing, both in academy as in industry. Consequently, the demand for machines with larger storage capacity and processing power constantly rises, as well as for tools that are able to handle such an amount of data by taking full advantage of the available resources. Parallel processing techniques are being widely used for building such tools [13], being Apache Hadoop one of the most prominent ones.

Apache Hadoop[2] is a framework that allows for distributed processing of huge amounts of data on computer clusters [14]. High scalability is one of its main features: from just a few to thousands of computer nodes can be employed for a single computation. High availability is also one of Hadoop's most important characteristics. Failure detection and handling are software-based and do not require special hardware.

Hadoop uses Map/Reduce, a relatively simple and nowadays very popular programming model [15]. In this model, the user must specify a *map* function that processes key/value pairs and produces sets of new pairs. Besides, a *reduce* function must also be defined. Such a function is applied to *map* results and aggregates all intermediate values associated to the same key, producing a final key/value pair for each one of them. Two important requirements are that these functions must be deterministic and must not have collateral effects. This way, they can be reapplied many times over the same initial dataset (if necessary) and produce the same result. This property allows for Hadoop to provide high-availability by re-executing operations performed by crashed nodes whose results were not propagated before their failures.

[2]Apache Hadoop Project, `http://hadoop.apache.org`, accessed September 21, 2016.

By using Hadoop, developers are isolated from issues inherent to parallel programming, *e.g.*, inter-process communication, concurrency control, race conditions and process failures. Hadoop takes charge of all these matters. Developers are even free from programming using the traditional OpenMP [16] and MPI [17] libraries, and dealing with their idiosyncrasies. Thus, they can focus on their original problems, trying to model a solution by using pairs of *map* and *reduce* functions written in Java or Python programming languages, for example.

Besides the Map/Reduce framework, Hadoop has another important component: the Hadoop Distributed File System (HDFS). Hadoop is designed to run on shared-nothing clusters, *i.e.*, clusters where each computer has its own processor(s), main memory and persistent storage device(s) (magnetic Hard-Disk Drives — HDD — or Solid-State Drives — SSD). HDFS uses all the available persistent storage devices to form a distributed, fault-tolerant file system that can be used by Map/Reduce applications to store data. Thus, the more devices are available, the more space can be used and, consequently, larger datasets can be processed. This approach eliminates the need for buying expensive, high-capacity storage devices, such as Storage Area Networks (SAN), making HDFS a very cost-effective alternative to them.

3. The Quandoop Simulator

Quandoop[3] is an open-source software that allows the simulations of discrete-time quantum walks on computer clusters. The software requirements are Hadoop version 1.2.1 and Java. Regarding hardware, Hadoop is designed to work on shared-nothing computer clusters. So, this is also a requirement for Quandoop.

In order to be consistent with the postulates of quantum mechanics, we must ensure that the state of the quantum walker at each step t is described by a unit vector $|\psi(t)\rangle$ on a Hilbert space, and that its evolution is described by a sequence of unitary operators $U_1, \ldots, U_n$ over that Hilbert space. Therefore, the classical simulation of discrete-time quantum walks consists on a sequence of matrix-vector multiplications, such as

$$|\psi(t)\rangle = U_n U_{n-1} \ldots U_1 |\psi(t-1)\rangle,$$

where t is a positive integer that represents the number of steps of the simulation, n is the number of unitary operators present in each step of

[3]The program can be downloaded from `https://github.com/david-ufrj/qw-hadoop/tree/master/Quandoop`.

the walker, $|\psi(t)\rangle$ is the state vector after t steps, and $|\psi(0)\rangle$ is the initial condition.

A simulation in Quandoop requires the necessary operators and state vectors to be stored in text files that are passed as input to the program. For each file, the first line must store its header and the remaining lines, the data, following this format:

- Header: `#TYPE,M,N`
  - `#`: identifies this line as the header.
  - `TYPE`: should be `A` or `B`, indicating whether the file stores the left or the right matrix to be multiplied, respectively.
  - `M,N`: number of rows and columns of the matrix, respectively.
  - Notice that even when the file is storing a state vector, its header must indicate a column matrix format, *i.e.*, `#TYPE,M,1`.
- Remaining lines: `TYPE,M,N,REALjIMAGINARY`
  - `TYPE`: should be `A` or `B`, indicating whether the file stores the left or the right matrix to be multiplied, respectively.
  - `M,N`: coordinates for row and column, respectively.
  - Notice that even when the file is storing a state vector, the coordinates for row and column must obey the format `TYPE,M,0`.
  - `REALjIMAGINARY`: real and imaginary parts of the corresponding entry of the matrix, and character `j` acting as a separator.

The simulator also requires the configuration file *config.properties*, with the following parameters that the user must provide before running the simulation:

- `steps`: number of steps to be executed in the simulation—must be an integer greater than zero.
- `paths`: path of the file that contains the paths to the folders of all matrices U and the vector $|\psi\rangle$. The order matters: it should be $U_0, \ldots, U_n, |\psi\rangle$, one per line. Every U file must be a matrix of type A and the ψ file must be a column matrix of type B.
- `workDir`: path in the HDFS where the program will store the data.
- `jarDir`: path of the directory where the files *quandoop.jar* and *operations.jar* are stored.

- `outputDir`: path of the directory where the result will be stored—all the files previously saved in that directory will be deleted after the simulation.
- `dimensions` (optional): The dimensions of the Hilbert subspaces—must be integer numbers greater than zero, and values must be separated by comma.
- `measurement` (optional): indices of the Hilbert subspaces which should be measured in the Pauli-Z observable—must be integers greater or equal to zero, and values must be separated by comma.
- `saveStates` (optional): saves partial states of the simulation—must be an integer greater than zero and less than `steps`, and indicates that the state vector must be saved after each step multiple of this value.

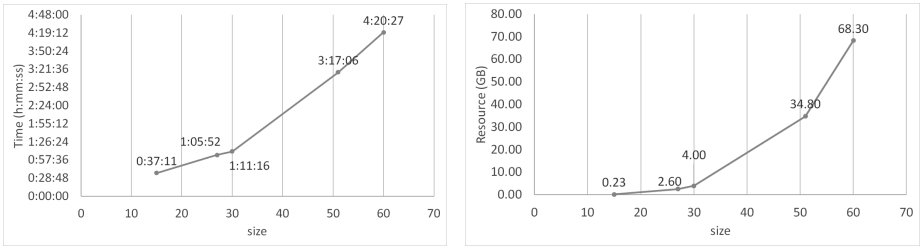
Once the required parameters are correctly filled and the input files are in the format accepted by Quandoop, the user can run the simulation, using this command line:

```
hadoop jar quandoop.jar quandoop.Quandoop
```

In the beginning, Quandoop copies the input files to the HDFS, so that they can be accessed during the simulation. Then, some variables are set according to the values passed by the user in the configuration file. After that, Quandoop performs the simulation and generates the results: complex amplitudes of the final state vector $|\psi(t)\rangle$, final probability distribution, norm, and partial states. These results are finally copied to the directory specified by the user. Temporary files created by Quandoop in HDFS are deleted after the simulation completion.

4. Experimental Evaluation

In this section, we describe a numerical experiment comprised of a one-dimensional quantum walk simulation with four interacting walkers. This experiment was motivated by the problem of molecular binding in interacting quantum walks [6]. As stated before, Quandoop simulator must receive input files with the operators needed to perform the simulation. For this purpose, we created a script to generate these input files with the values for our simulation. This script has a variable (*size*) used to generate different input files to Quandoop. The *size*, in this case, refers to the number of vertices of the graph in which the walk takes place.



(a) (b)

Figure 1: Results from simulations on the computer cluster. (a) Elapsed time. (b) Disk usage.

Using a sequential code to perform this simulation on a workstation with an Intel XEON E5-1620 processor (3.6 GHz) and 28 GB RAM, we can only increase the value of the variable *size* until 51. Beyond that value, the program execution crashes due to memory overflow. However, with Quandoop, we can easily surpass this value, because the operators used in the simulation are stored on HDFS, which has a much higher capacity when compared to RAM.

For running the parallel simulation version, we employed a 32-node cluster with shared-nothing architecture. One node was used as frontend[4]. All the remaining nodes (thirty one) were used for running Hadoop. Their setup is as follows: one Intel Core i7 3770 (3.4 GHz) processor, 8 GB RAM and 1 TB HDD. They are interconnected through a Gigabit network.

Fig.1(a) shows the time variation to perform this simulation using Quandoop for different values of the variable *size*. Fig.1(b) shows the amount of storage needed to perform the simulation for different values of *size*.

Due to time restrictions related to the cluster utilization, we were able to increase *size* value only until 60. This is the only reason why we went no further. No RAM overflow occurred because Quandoop employs HDFS to store the operators. Fig.1(b) shows that the maximum disk space used is 68.3 GB, far below our HDFS total capacity, comprised of thirty-one 1 TB disks. Our results show that Quandoop has the potential to perform simulations with much more graph vertices than those achieved with memory-only tools. We could have done that if no time usage restrictions were imposed. Besides, it is easier and less expensive to augment total disk's storage capacity than RAM's. To augment RAM size, one needs free slots on the

[4]Server node that manages the entire cluster, controlling, monitoring and distributing tasks to the other nodes.

computer's motherboard and the resulting expansion is usually limited to a few gigabytes. Adding a 1 TB disk to a computer is easier and less expensive, besides providing more storage space. We conclude that Quandoop has the potential to help quantum walk researchers to perform simulations in classical computers far beyond the limits imposed by sequential, in-memory solutions. The more cluster nodes available, the more graph vertices can be simulated. Hadoop makes even possible to use more than one cluster, significantly expanding Quandoop limits.

5. Conclusion

In this paper, we described Quandoop, a classical simulator of discrete-time quantum walks on computer clusters. The main advantage of Quandoop over previous solutions is that our software supports simulations with very large memory requirements even with a low cost infrastructure. The use of Hadoop, a popular Big Data processing framework, and its Map/Reduce programming model makes it possible. Although Quandoop focus on the simulation of quantum walks, other oracular quantum algorithms could also be simulated with small adaptations.

The simulator was tested on a computer cluster composed by 32 low cost nodes (including the frontend), each of them equipped with processor Intel Core i7 3770, 3.4 Ghz, 8 GB RAM, 1 TB HDD. They were interconnected through a Gigabit network. Due to time restrictions related to the cluster utilization, we were able to simulate quantum walks on graphs increasing the number of vertices until 60. However, no RAM overflow occurred and the maximum disk space used was far below our HDFS total capacity, indicating that Quandoop could still simulate much larger systems. On the other hand, when we used a sequential code to perform the same tests on a computer with 28 GB RAM, we could only increase the number of vertices until 51. Beyond that value, the program execution crashed due to memory overflow.

We conclude that Quandoop has the potential to help researchers to simulate quantum walks far beyond the limits imposed by the current approaches, which may have impacts on the design and analysis of quantum algorithms as well as in the study of interesting physical phenomena.

As a future work, Quandoop should be used on a computer cluster without time restrictions in order to test its limits. The interface with other simulators—such as HiPerWalk or QWalk—should also be further developed in order to become more user-friendly. Furthermore, we are working on replacing Hadoop by Apache Spark[5], another Big Data processing framework

[5]Apache Spark Project, `http://spark.apache.org/`, accessed September 23, 2016.

that extends Hadoop computation model and presents better performance. A very well-known Hadoop characteristic that limits its performance is the need for always storing intermediate results on disks. Spark does not present this limitation, being able to process these results directly in main memory. This feature significantly improves its performance when compared to Hadoop, besides providing a better usage of the available resources. With Spark, we believe graphs with more than 60 vertices could have been simulated even with the time restrictions we faced in the experiments described in this work.

Acknowledgments. The authors thank Renato Portugal and Ricardo Farias for helpful discussions. This work was supported by CNPq, CAPES and FAPERJ.

References

[1] Ambainis, A., SIAM Journal on Computing **37** (2007) 210.

[2] Magniez, F., Santha, M., and Szegedy, M., SIAM Journal on Computing **37** (2007) 413.

[3] Farhi, E., Goldstone, J., and Gutmann, S., Theory of Computing **4** (2008) 169.

[4] Childs, A. M., Gosset, D., and Webb, Z., Science **339** (2013).

[5] Romanelli, A., Donangelo, R., Portugal, R., and Marquezino, F. d. L., Physical Review A **90** (2014) 022329.

[6] Ahlbrecht, A. et al., New Journal of Physics **14** (2012) 073050.

[7] Marquezino, F. and Portugal, R., Computer Physics Communications **179** (2008) 359.

[8] Lara, P., Leao, A., and Portugal, R., Simulation of quantum walks using HPC, in *Proceedings of the 3rd Conference of Computational Interdisciplinary Sciences*, pages 230–242, Asunción, Paraguay, 2014, Pan American Association of Computational Interdisciplinary Sciences.

[9] Aharonov, Y., Davidovich, L., and Zagury, N., Physical Review A **48** (1993) 1687.

[10] Farhi, E. and Gutmann, S., Phys. Rev. A **58** (1998) 915.

[11] Portugal, R., Physical Review A - Atomic, Molecular, and Optical Physics **93** (2016).

[12] Szegedy, M., Quantum speed-up of markov chain based algorithms, in *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 32–41, IEEE, 2004.

[13] Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D., and Moon, B., ACM SIGMOD Record **40** (2011) 11.

[14] White, T., *Hadoop: The definitive guide*, O'Reilly Media, Inc., Massachusetts, 3 edition, 2012.

[15] Dean, J. and Ghemawat, S., Communications of the ACM **51** (2008) 107.

[16] Chapman, B., Jost, G., and Pas, R. V. D., *Using OpenMP: portable shared memory parallel programming*, MIT press, Cambridge, MA, 2008.

[17] Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: portable parallel programming with the message-passing interface*, MIT press, Cambridge, MA, 1 edition, 1999.