InterGridSim: A Broker-Overlay Based Inter-Grid Simulator

Abdulrahman Azab(✉)

Division of Research Computing, University Center for Information Technology, University of Oslo, Oslo, Norway
azab@uio.no

Abstract. Large scale Grid computing systems are often organized as an inter-Grid architecture, where multiple Grid domains are interconnected through their local broker. In this context, the main challenge is to devise appropriate job scheduling policies that can satisfy goals such as global load balancing together with maintaining the local policies of the different Grids. This paper presents InterGridSim, a simulator for scalable resource discovery and job scheduling technique in broker based interconnected Grid domains. Inter-Grid scheduling decisions are handled jointly by brokers in a broker overlay network. A Broker periodically exchanges its local domain's resource information with its neighboring brokers. InterGridSim offers several network structures and workload allocation techniques for Tier-1 and Tier-0 networks and large workload capacity. The paper presents sample simulations for throughput, utilisation, and load balancing in a network of 512 brokers and 50k nodes.

1 Introduction

Grid computing is based on coordinated resource sharing in a dynamic environment of multi-institutional virtual organizations, VOs [1]. The target of computational Grid, which is our main focus, is to aggregate many Grid compute resources as one powerful unit on which computational intensive applications can run and produce results with low latency. Computational Grid Model is mainly composed of three components: (i) worker/ executor, to which computational jobs are submitted and where they are executed, (ii) client/ user, from which jobs are submitted and by which the Grid is consumed, and (iii) broker/ scheduler, which is responsible of allocating submitted jobs by clients to suitable workers [2]. The InterGrid concept [3] has been evolved due to the dramatic increase in the resource demands of Grid application together with the submission rate. The idea of resource sharing between different domains is already in use in the network level and known as peering [4–6]. The interconnection of Grid domains may be implemented in one of three levels:

Client level where the client/user machine can have access to multiple Grid domains using associated access rights [7,8]. This can be implemented either by granting multiple access rights to each Grid client [7], or by installing multiple Grid clients on the same user machine to access multiple domains with different

M. Qiu et al. (Eds.): SmartCom 2021, LNCS 13202, pp. 374–383, 2022.
https://doi.org/10.1007/978-3-030-97774-0_34

architectures [8]. This alternative is not scalable, since it is not applicable to grant access to hundreds of domains to thousands of clients which may result in a massive number of contentions.

Worker level where worker/executer nodes could have the task executors of multiple Grid domains installed so that it become available for submission requests from either of those domains [9]. Based on our experience [10], this alternative would negatively influence the capacity of the worker machine which would in turn have a negative result on the resource capacity in each of the interconnected domains.

Broker level where the interconnection is to be carried out through *Local Resource Brokers*, LRB. Two methodologies have been implemented in this direction: (i) *Central meta-scheduler*, and (ii) *Grid federation*. The role of a central meta-scheduler [11] to manage and control the interGrid submission requests allocating each to a LRB with matching resource requirements on its domain. This methodology is implemented by Condor-G [12] where the Condor-G meta-scheduler can exchange submission requests between Condor pools and Globus VOs. A similar mechanism is implemented by Nimrod-G [13]. This methodology suffers from the centralization problem where the meta-scheduler may be overloaded with inter-Grid requests, in addition to single point of failure. Grid federation is to establish the interconnection between LRBs in an overlay, giving equal rights to all connected brokers to participate in the interconnection task allocation decision. Such a federation of Grid domains [3] would avoid the limitations of the central meta-scheduler methodology. This methodology is implemented in condor-flock-p2p [14] through the establishment of a pastry [15] based p2p overlay between brokers. A little different mechanism is adopted by the InterGrid project [3,16] where LRBs are responsible only for local brokering while the interconnection and management of interGrid submission requests are carried out by fixing a dedicated gateway in each domain.

This paper presents InterGridSim [17], a simulator for Inter-Grid resource management. Different techniques can be implemented in InterGridSim from fully centralised to peer-to-peer. The main technique promoted in InterGridSim is Slick [18–20] which is built on a hybrid peer-to-peer overlay network [21]. Slick aims at reducing the overall complexity of the system, enabling transparent access to regular participants, while ensuring efficient resource utilization, load balancing and failure handling. The underlying idea of the architecture is that each participating node may offer or claim computational resources as necessary for their application. This technique is suitable for interconnected domains, each with one broker node responsible for local resource management within its virtual organization. The broker receives requests for resources from participating nodes, compare the requirements in each request with available resources at nodes in the network, and forwards the requests to suitable nodes. Each node interacts only with its attached broker, and both regular node and broker failures are handled. Brokers associated with the different domains take part in an overlay network of brokers that are responsible for global resource management and task deployment throughout the network.

InterGridSim has been implemented in the PeerSim simulation environment [22], and has been evaluated experimentally for various load conditions, network sizes, and topologies. The results show that the architecture is able to allocate compute tasks quickly and efficiently for different broker overlay topologies.

2 The Inter-grid Architecture

The Inter-Grid architecture in InterGridSim is based on global resource sharing and collaboration of Grid domains. Each domain consists of one domain controller (i.e. Broker), and a collection of regular nodes. Components of the grid system are:

Job in InterGridSim refers to a computational job. It has five execution parameters: 1) Required CPU, the computational power required for running the job. 2) Required Memory, the memory size required for running the job. 3) Expiration Time, the amount of time to wait for allocation. 4) Creation Time, the time at which the job is created for allocation. 5) Allocation attempts, the maximum number of attempts to deploy the job before it is expired.
Regular node refers to each non-broker node in the Grid. Each regular node can be a member of one domain, and can submit and/or run a job. A regular node is also responsible for periodically sending information about the current available resource state of the node to its broker. Each regular node has two resource parameters: 1) Available CPU, which refers to the available computational power in the node, and 2) Available Memory space. Regular is equivalent to Peer in HIMAN, which contains two components: Worker (W), which is responsible for task execution, and Client (C), which is responsible for task submission [23,24].
Broker is a node which works as a domain controller, can also work as a regular node in case of lack of available regular nodes. It is responsible for: 1) Allocating jobs to suitable nodes. A suitable node for a job is elected by performing a matchmaking process between the job requirements and the available resources of attached Grid nodes [2]. 2) Storing the current resource state for local nodes (i.e. in the same domain) as well as global nodes (i.e. in other domains).
Grid Domain (Virtual Organization) is an overlay of nodes, which can be allocated in different regions and be members of several organizations. Each domain is composed of one broker and regular nodes and is structured as a star logical topology, so that; communication is between the broker and regular nodes. There is no direct communication between regular nodes within the same domain.
Broker overlay is a network of brokers through which communication and data exchange between different Grid domains is performed.

InterGridSim simulates resource discovery and global job scheduling for interconnected Grid domains. InterGridSim supports several architecture of the broker overlay. One is structured-p2p [15] that each broker has a `nodeID` and a routing table, and the routing table of each broker is filled with `nodeID`s of brokers which share different prefixes with the current broker's `nodeID`. Another

example is gossip [25] where each broker has a set of neighbors, and resource information is distributed through periodically exchanging data with a neighbor broker. Slick, implements the first architecture where each broker must be holding a routing table in which the addresses of its neighboring brokers are stored. InterGridSim gateway broker is designed to work on the top of the local broker of the Grid domain as three layers architecture. The different layers and components of Slick are described in the main Slick paper [18].

Fault Tolerance: InterGridSim mainly manages Broker failures, where worker and client failures are managed internally by the broker in each Grid domain. Each regular node has direct communication with its broker. Periodically, each node sends its resource information to the broker to update its associated resource-information record to the current state. Each node holds a list of information about all existing brokers in the broker overlay. This information is retrieved and updated periodically from its local broker. A regular detects its local broker failure when it attempts to send its resource information to the broker. In case of broker failure, all regular nodes in the domain are detached from the Grid, and each node sends a membership request to the first broker in the list. If the request is granted, the regular node sets the new broker as the attached broker; otherwise the request is repeated to the next broker in the list.

3 Simulation Model

InterGridSim is designed using PeerSim [22]; a Java-based simulation-engine designed to help protocol designers in simulating their P2P protocols. The simulation model is based on cycle-based simulation. Input parameters for the simulation engine are read from a configuration text file. In cycle-based simulation, each simulation cycle is considered as one time unit. Four main Interfaces are used: `Node`, `Linkable`, `CDProtocol`, and `Control`. The overlay network is a collection of `Node` objects. Before starting simulation, a collection of `Initializer` objects, specified in the configuration file, are created and execute initialization functions. All `Initializer` objects must implement `Control` Interface This initialization process includes constructing the network by connecting `Node` objects together based on the specified topology. Pointers to all neighboring nodes are stored in a `Linkable Protocol` object attached to each `Node` object. Any other initializations can be included. The default `Linkable Protocol` is the `IdleProtocol`. Each node object is attached to a collection of `CDProtocol` (i.e. cycle driven protocol) objects. Each `CDProtocol` object is responsible for simulating one communication protocol in the attached node with identical objects in other nodes. This is carried out by calling a `nextCycle()` method in each `CDProtocol` object by the simulation engine each simulation cycle. Each simulation cycle, the simulation engine calls a collection of `execute()` methods in `Control` objects. `Control` objects are created to carry out all control operations needed for the simulation, including modification of simulation parameters. Another role of `Control`

objects is the observation and recording of data related to simulation environment state each simulation cycle. All Control objects must implement `Control` Interface.

In this model, a `GridNode` class implements the `Node` Interface is built. `GridDeployer`, and `GridFailureControl` class objects are included as a `Control` object for performing job deployment and failure handling. Three CD Protocols as `CDProtocol` classes are built:

Grid CD Protocol. This protocol is included in each regular node and responsible for communicating with the attached broker and sends the resource information each simulation cycle.

Deployment Protocol. This protocol is included in each regular node. It is responsible for responding to the deployment requests from the broker by one of two responses: *Deployed*, if the job deployment is successful, and *Failed*, if the local resources are not enough for deploying the job.

Grid Broker Protocol. This protocol is included in broker nodes and responsible for: 1) Receiving jobs from the job deployer and append them to the job queue. 2) Receiving resource information from attached regular nodes and replaces the current stored blocks in the resource information data set with the new ones. 3) Picking one job each cycle from the job queue and invoking the job deployment algorithm. 4) Exchanging resource information with one neighbor broker each cycle by invoking the resource information exchange algorithm.

Figure 1 describes the Grid simulation model and the communication between different protocols. `GridNode` class is a reference for node objects. `GridAllocator` and `GridFailureControl` classes are included as references for `Control` objects which simulate job allocation and failure handling. Three cycle-driven `Protocol` classes are also built: 1) `Grid CD Protocol`, included in each

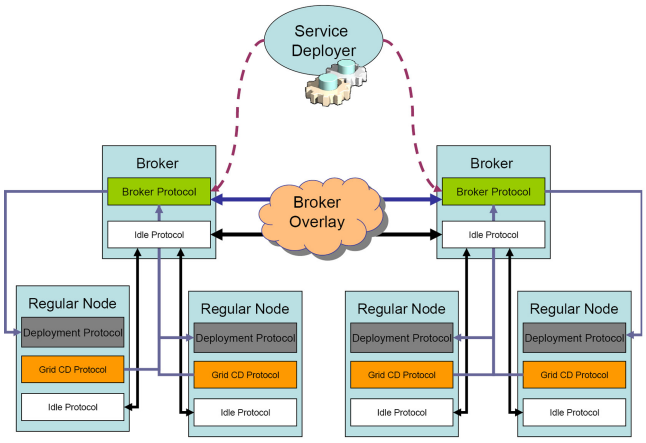


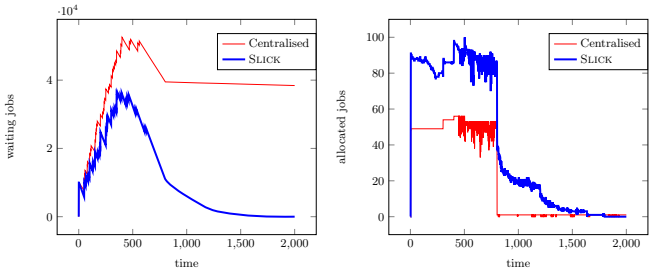
Fig. 1. Simulation model

regular node and is responsible for communicating with the attached broker and sends the resource information in each simulation cycle. 2) `Allocation Protocol`, included in each regular node and is responsible for responding to the Allocation requests from the broker. 3) `Grid Broker Protocol`, included in each broker node for performing the tasks associated with the broker (described in the previous sections). The `Idle Protocol` is in the main `PeerSim` package and is included in each node to be responsible for establishing communication with neighboring nodes.

4 Simulation Results

We present the results of simulating a large number of domains with InterGridSim inter-Grid simulator using Slick workload management technique. We simulate a system of 50,000 nodes in 512 interconnected domains. The domains are connected through local brokers in a HyperCube logical topology, i.e. in case of a network size of N, each broker will have k neighbours in its routing table where $k = \frac{\ln N}{\ln 2}$. In case of 512 brokers, each broker will have 9 neighbors. Slick is tested against the centralised meta-scheduling technique where we implement logical star topology between an orchestrator and the brokers in the broker overlay. Compute node specifications are of two groups which are different in four static attributes: [group1: 2 CPU slots, 4 GB Memory, Windows OS, No java support] and [group2: 4 CPU slots, 8 GB Memory, Linux OS, Java support] Nodes are divided equally between the two groups, 25,000 each, but scattered among the domains. We create a load of total 80,000 synthetic jobs divided into 100 sequences. Each sequence is assigned to one broker. Using a random frequency $50 < f < 100$ time instance, a random number of jobs $50 < j < 100$ is submitted periodically by each sequence. Job resource requirements are randomly set. The process continues until all the 80,000 jobs are submitted. The total simulation time of the experiment is set to 2000 time instances. Each time instance, the local scheduler processes one job from the local queue, and the gateway scheduler processes one job from the gateway queue. Each time instance each broker synchronizes the resource information database with one neighbor broker.

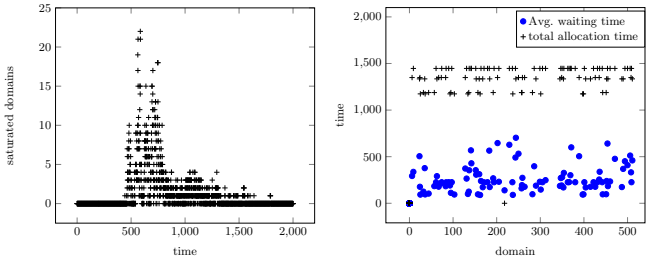
We use three benchmarks: *Job allocation throughput*, *resource utilisation*, and *Load balancing*. Job allocation throughput is measured by reading the total number of waiting jobs in the system/time, Fig. 2(a), and number of job allocations/time Fig. 2(b). It is clear that Slick is achieving higher throughput. Slick manages to reach a steady state were all jobs are allocated, within 1344 time instances, while in other systems, a bottleneck case happens. This can be described that in case of centralized allocation, there is only the central meta-scheduler to carryout the interconnections, which in case of cross-domain submissions allocates only one job per time instance. The breakdown both cases is after $\approx$ 800 time instances is because all job sequences complete their submissions by that time. We made this in purpose in order to validate the system performance when job allocation is carried out only inter-domain and not intra-domain. Resource utilisation is measured by reading the number of saturated



(a) Total waiting jobs (b) Allocated jobs per time instance

Fig. 2. System throughput: overall job allocation ratio

domains/time, e.g. those domains which workers are fully saturated with jobs. Figure 3(a) shows that SLICK in the time of high load ≈ 500 time instances, is achieving larger utilisation. Load balancing is measured by calculating for brokers, throughout the simulation: How long did it take to allocate all jobs owned by the domain of each broker, and what is the average waiting time. In Fig. 3(b), it is clear that for SLICK, the total allocation time never exceeded 1500 time instances and the maximum average waiting time is below 800. For centralized allocation, none of the domains got all of its jobs allocated during the 2000 time. The value of 2000 for both total allocation time and average waiting time indicates that this broker's jobs were not totally allocated.



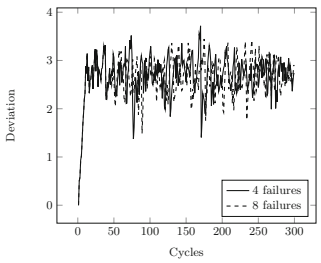
(a) System utilisation: Number of saturated domains per unit time (b) Load balancing: Average waiting time and total allocation time of jobs submitted to each of 100 domains

Fig. 3. Broker overlay coordination with 100 domains

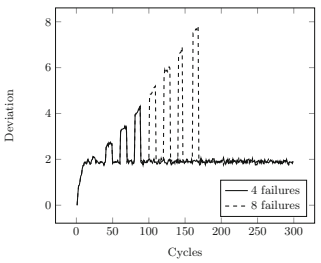
Fault Tolerance: This experiment demonstrates how the broker overlay based architecture is tolerant to broker failures. Broker failures are injected during the simulation. With the existence of broker failures, it is expected that the deviation of the reading time values of the stored resource information from the current

cycle will increase due to failure. The reason is that resource information of the regular nodes which have been attached to the failed broker, will remain old and not updated until they are attached to other brokers and start sending resource information blocks. In the following experiments, a new parameter is taken into account: *Data Age*, which is the maximum age in cycles of resource information in a broker resource data set. In each simulation cycle, the broker protocol checks the reading time of each block in the resource information data set. If the reading time of a block is $\leq (Currenttime - DataAge)$, then, this block is removed from the data set. If a new block for the same node is received later, in an exchange operation, it is added to the data set. The following experiments are performed by varying the value of Data Age.

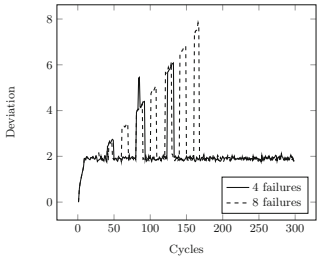
Four topologies are used: ring, fully connected, and Wire-k-Out ($k = 60$), and hyper-cube. The network size is fixed to N = 500, and M = 100. The number of simulation cycles is 300. The experiment is performed with varying the number of broker failures: The data age is fixed to 10 cycles with 4 and 8 injected broker failures, depicted in Fig. 4.



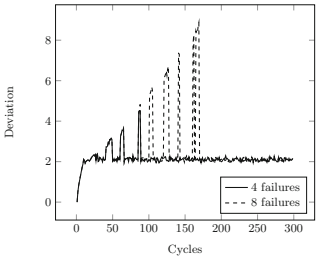
(a) Ring broker overlay topology



(b) Fully Connected broker overlay topology



(c) Wire-k-Out broker overlay topology, $k = 60$



(d) Hyper-cube broker overlay topology

Fig. 4. Impact of failures on the deviation of the resource information for: data age of 10 cycles with 4 and 8 injected broker failures

In Fig. 4, it is clear that for fully connected, wire-k-out, and hyper-cube topologies, the system can recover from failures and return to stable state. But When Data Age = 30, the system stability is not settled because of the existence of old data. In case of ring topology, the deviation has terrible and unstable variation with failures. This can be described that, because of the lack of possible direct communications between brokers, it takes time for a broker to reach data stored in non-neighbor brokers.

5 Conclusions and Acknowledgments

This paper presented InterGridSim, a simulator for interconnected Grid domains. The key feature of InterGridSim is that both resource state and hardware specifications of each domain are stored in small datasets which enables the exchange of resource information among brokers. Using this information in matchmaking, cross-scheduling decisions are made accurate in most cases. InterGridSim offers several network structures and workload allocation techniques and large workload capacity. The paper presented sample simulations for throughput, utilisation, and load balancing in a network of 512 brokers and 50k nodes.

InterGridSim development has been partially funded by NeIC (Nordic e-Infrastructure Collaboration) [26] for supporting the development of Nordic Tier-1 activity [27]. InterGridSim has also been developed as part of WP5 in the EOSC-Nordic project [28]. EOSC-Nordic has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652.

References

1. Joseph, J., Fellenstein, C.: Grid Computing, vol. 1. Prentice Hall Professional, Upper saddle River (2004)
2. Raman, R., Livny, M., Solomon, M.: Matchmaking: distributed resource management for high throughput computing. In: Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7), Chicago, IL, July 1998 (1998)
3. Assuno, M.D.D., Buyya, R., Venugopal, S.: Intergrid: a case for internetworking islands of grids. In: Concurrency and Computation: Practice and Experience (CCPE), pp. 997–1024 (2007)
4. Baake, P., Wichmann, T.: On the economics of internet peering. NETNOMICS **1**, 89–105 (1999). https://doi.org/10.1023/A:1011449721395
5. Chakrabarti, S., Badasyan, N.: Private peering, transit and traffic diversion. NETNOMICS **7**(2), 115–124 (2005). https://doi.org/10.1007/s11066-006-9007-x
6. Huston, G.: Interconnection, peering and settlements-part I. Internet Protocol. **2**(1) (1999)
7. Evers, X., de Jongh, J.F.C.M., Boontje, R., Epema, D.H.J., van Dantzig, R.: Condor flocking: load sharing between pools of workstations. Technical report, Delft, The Netherlands (1993)

8. Aiftimiei, C., et al.: Design and implementation of the gLite CREAM job management service. Future Gener. Comput. Syst. **26**(4), 654–667 (2010)
9. NorduGrid: Nordic Testbed for Wide Area Computing and Data Handling. http://www.nordugrid.org/
10. Azab, A., Meling, H.: Stroll: a universal filesystem-based interface for seamless task deployment in grid computing. In: Göschka, K.M., Haridi, S. (eds.) DAIS 2012. LNCS, vol. 7272, pp. 162–176. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30823-9_14
11. Schopf, J.: Ten actions when superscheduling. In: Global Grid Forum (2001)
12. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-G: a computation management agent for multi-institutional grids. Clust. Comput. **5**(3), 237–246 (2002). https://doi.org/10.1023/A:1015617019423
13. Buyya, R., Abramson, D., Giddy, J.: Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid. In: Proceedings of HPC ASIA 2000, pp. 283–289 (2000)
14. Butt, A.R., Zhang, R., Hu, Y.C.: A self-organizing flock of condors. J. Parallel Distrib. Comput. **66**(1), 145–161 (2006)
15. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45518-3_18
16. Assuncao, M.: Provisioning techniques and policies for resource sharing between grids. Ph.D. dissertation, The University of Melbourne, Australia (2009)
17. Azab, A.: Intergridsim: inter-grid simulator based on peersim. https://github.com/abdulrahmanazab/intergridsim
18. Azab, A., Meling, H.: Slick: a coordinated job allocation technique for inter-grid architectures. In: 7th European Modelling Symposium (EMS) (2013)
19. Azab, A., Meling, H., Davidrajuh, R.: A fuzzy-logic based coordinated scheduling technique for inter-grid architectures. In: Magoutis, K., Pietzuch, P. (eds.) DAIS 2014. LNCS, vol. 8460, pp. 171–185. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43352-2_14
20. Azab, A.: Binary matchmaking for inter-grid job scheduling. In: Silhavy, R., Senkerik, R., Oplatkova, Z.K., Silhavy, P., Prokopova, Z. (eds.) Modern Trends and Techniques in Computer Science. AISC, vol. 285, pp. 433–443. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06740-7_36
21. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. **36**(4), 335–371 (2004)
22. Montresor, A., Jelasity, M.: PeerSim: a scalable P2P simulator. In: Proceedings of the 9th International Conference on Peer-to-Peer (P2P'09), Seattle, WA, Sep 2009, pp. 99–100 (2009)
23. Condor project. http://www.cs.wisc.edu/condor/
24. El-Desoky, A.E., Ali, H.A., Azab, A.A.: A pure peer-to-peer desktop grid framework with efficient fault tolerance. In: ICCES'07, Cairo, Egypt, pp. 346–352 (2007)
25. Allavena, A., Demers, A., Hopcroft, J.E.: Correctness of a gossip based membership protocol. In: Aguilera, M.K., Aspnes, J. (eds.) PODC, ACM, pp. 292–301 (2005)
26. Neic: Nordic e-infrastructure collaboration. https://neic.no/
27. Nordic wlcg tier-1 facility. https://neic.no/nt1/
28. Eosc-nordic project. https://eosc-nordic.eu/