Subordination:
Cluster Management without Distributed Consensus

Ivan Gankevich, Yuri Tipikin
Dept. of Computer Modeling and Multiprocessor Systems
Saint Petersburg State University
Saint Petersburg, Russia
igankevich@yandex.com, yuriitipikin@gmail.com

Vladimir Gaiduchok
Dept. of Computer Science and Engineering
Saint Petersburg Electrotechnical University "LETI"
Saint Petersburg, Russia
gvladimiru@gmail.com

POSTER PAPER

***Abstract*—Nowadays, many cluster management systems rely on distributed consensus algorithms to elect a leader that orchestrates subordinate nodes. Contrary to these studies we propose consensus-free algorithm that arranges cluster nodes into multiple levels of subordination. The algorithm structures IP address range of cluster network so that each node has ranked list of candidates, from which it chooses a leader. The results show that this approach easily scales to a large number of nodes due to its asynchronous nature, and enables fast recovery from node failures as they occur only on one level of hierarchy. Multiple levels of subordination are useful for efficiently collecting monitoring and accounting data from large number of nodes, and for scheduling general-purpose tasks on a cluster.**

***Keywords*—job scheduling, leader election, cluster monitoring, cluster accounting, cluster management**

I. INTRODUCTION

Many distributed systems rely on a leader that manages computer cluster and schedules jobs on other nodes. The leader role can be assigned either statically – to a particular physical node – or dynamically through election process. In the first case fault-tolerance is achieved by duplicating leader functionality on some reserved node which is used as a substitute in case of current leader failure. In the second case, fault-tolerance is guaranteed by re-electing failed leader by survived nodes. Although, dynamic election requires dedicated distributed algorithm, this approach becomes more and more popular as it does not require spare nodes to survive leader failure.

Leader election algorithms (which are sometimes referred to as *distributed consensus* algorithms) are special cases of wave algorithms. In [1] Tel defines them as algorithms in which termination event is preceded by at least one event occurring in each parallel process. Wave algorithms are not defined for anonymous networks, that is they does not apply to processes that can not uniquely define themselves. However, the number of processes can be determined in the course of an algorithm. In distributed system this means that wave algorithms work for clusters with dynamically changing number of nodes, and nodes can go online and offline.

Contrary to this, our approach does not use wave algorithms, and hence does not require communicating with each node in a cluster to determine a leader. Instead, each node enumerates all nodes in the network it is part of, and converts this enumeration to a tree with a user-defined maximal fan-out value. Then the node determines its level of hierarchy and tries to communicate with nodes from higher levels to become subordinate. First, it checks the closest ones and then goes all the way to the top. If it can not find any top-level node, it becomes the root node.

Enumeration of all hosts in a network defines strict total order on a set of cluster nodes. Although, technically any function can be chosen to map a node to a number, in practise this function should be sufficiently smooth and may have infrequent jumps. For example, high-frequency oscillations (e.g. representing measurement error), may result in rapid changes in the hierarchy of nodes, which make it useless for end users. Thus, any such peculiarities should be smoothed in order to make mapping useful. The ideal mapping function is a line, which represents static node mapping.

Smoothness of the map function ensures stability of hierarchy of nodes and minimises overheads of hierarchy changes. The simpliest such function is the position of an IP address in network IP address range: a line which jumps only when network configuration is changed (which a rare occasion).

So, the main feature of this algorithm is relation of subordination. It makes hierarchy updates local to a particular level and branch of a hierarchy, and allows precise determining of the leader of each node.

II. RELATED WORK

One point that distinguishes our approach with respect to some existing proposals [2]–[4] is that our algorithm elects multiple leaders thus creating *leaders' framework* with multiple levels of subordination. The topology of this framework

reflects the topology of underlying physical network as mush as possible, however, if the number of nodes connected to a single switch is large, additional levels of subordination may be introduced.

In contrast to many leader election algorithms, our algorithm is not intended to manage updates to some distributed database. The main application of leader framework is to distribute workload across large number of luster nodes. Typically one cluster is managed by one master server (possibly by two servers to improve fault tolerance), which collects monitoring and accounting data, issues cluster-wide configuration commands, and launches jobs. When the cluster becomes large, master server may not cope with the load. In this case, introducing subordinate servers solves the issue.

The method described in the following sections relies on node performance and node latency (for geographically distributed clusters) being stable. Otherwise the method produces randomly changing framework of leaders which does not allow to distribute the load efficiently. For local clusters the method is always stable as the leader is determined by its position in Internet IP address range which rarely changes.

To summarise, the method described in the following sections may not be suitable for managing updates to a distributed database, and for environments where IP addresses are changed frequently. Its sole purpose is to distribute the load on the cluster with large number of nodes.

III. METHODS

A. NODE MAPPING

Relation of subordination can be defined as follows. Let $\mathcal{N}$ be the set of cluster nodes connected to a network, then

$$\forall n_1 \forall n_2 \in \mathcal{N}, \forall f : \mathcal{N} \to \mathcal{R}^n \\ \Rightarrow (f(n_1) < f(n_2) \Leftrightarrow \neg(f(n_1) \geq f(n_2))),$$

where f maps a node to a number and operator $<$ defines strict total order on $\mathcal{R}^n$. Thus, f is the function that defines node's rank allowing to compare nodes to each other. and $<$ is binary relation of subordination which ensures that the mapping is bijective.

The simpliest function f maps each node to its Internet address position in network IP address range. Without conversion to a tree – when only *one* leader is allowed in the network – a node with the lowest position in this range becomes the leader. If a node occupies the first position in the range, then there is no leader for it. Although, IP address mapping is simple to implement, it introduces artificial dependence of the leader role on the address of a node. Still, it is useful for initial configuration of a cluster when more complex mappings are not possible.

The more sophisticated mapping may use performance to rank nodes in a network. Sometimes it is difficult to determine performance of a computer: The same computers may show varying performance for different applications, and the same applications may show varying performance for different computers. In this work performance is estimated as the number of jobs completed per unit of time (which makes it dependent on the type of a workload).

Several nodes may have the same performance, so using it as function f may violate strict total order. To implement performance-wise rankings the two mappings can be combined into a compound one:

$$f(n) = \langle 1/\text{perf}(n), \text{ip}(n) \rangle.$$

Here *perf* is performance of a node estimated as the number of jobs completed per unit of time, and *ip* is the mapping of a node to its Internet position in network IP address range. So, with the compound mapping each node is characterised by both its performance and position in the network address range. When performance data is available, it supersedes node's position in the network for ranking.

For a cluster with *linear* topology (all computers connected to a switch) knowing performance is enough to rank nodes, however, for a distributed system network latency may become a more important metric. For example, a high-performance node in a distant cluster may not be the best choice for a leader if there are some intermediate nodes on the way to it. Moreover, network latency is a function of at least two nodes, and using it in the mapping makes it dependent on the node which produced it. Thus, each node in the cluster has its own ranked list of candidates, and a node may occupy a different position in each list. Since, each node has its own ranked list, multiple leaders are possible within network. Finally, measuring network latency for each node introduces substantial overhead which can be minimised with conversion to a tree (see Section III-B).

Although, putting network latency into the mapping along with other metrics seems feasible, some works suggest that programme speedup depends on its ratio to performance. For example, in [5]–[7] the authors suggest generalisation of Amdahl's law formula for computer clusters. The formula

$$S_N = \frac{N}{1 - \alpha + \alpha N + \beta\gamma N^3}$$

shows speedup of a parallel programme on a cluster taking into account communication overhead. Here N is the number of nodes, α is the parallel portion of a program, β is the diameter of a system (the maximal number of intermediate nodes a message passes through when any two nodes communicate), and γ is the ratio of node performance to network link performance. Speedup reaches its maximal value at $N = \sqrt[3]{(1-\alpha)/(2\beta\gamma)}$, so the higher the link performance is the higher speedup is achieved. This particular statement ratifies the use of node performance to network latency ratio in the mapping, so that final mapping is as the following.

$$f(n) = \langle \text{lat}(n)/\text{perf}(n), \text{ip}(n) \rangle,$$

where *lat* is measured network latency between the node which composes ranked list and the current node in the list.

So, putting network latency to the mapping unnecessary complexifies configuration of local cluster, but can be beneficial for cluster federation. In case of local homogeneous cluster an IP address mapping should be enough to determine the leader.

B. SUBORDINATION TREE

To make leader election algorithm scale to a large number of nodes, enumeration of nodes is converted to a tree. In the tree each node is uniquely identified by its level l and offset o, which are computed as follows.

$$l(n) = \lfloor \log_p n \rfloor,$$
$$o(n) = n - p^{l(n)},$$

where n is the position of node's IP address in network IP address range, and p is the maximal number of subordinate nodes. The leader of a node with level l and offset o has level $l-1$ and offset $\lfloor o/p \rfloor$. The distance between any two nodes in the tree with network positions i and j is computed as

$$\langle \text{lsub}(l(j), l(i)), |o(j) - o(i)| \rangle,$$
$$\text{lsub}(l_1, l_2) = \begin{cases} \infty & \text{if } l_1 \geq l_2, \\ l_1 - l_2 & \text{otherwise.} \end{cases}$$

The distance is compound to make level dominant.

To determine the leader a node ranks all nodes in the network according to mapping $\langle l(\text{ip}(n)), o(\text{ip}(n)) \rangle$, and using distance formula chooses the node which is closest to computed leader's position and has lower position in the ranking. That way offline nodes are skipped, however, for sparse networks (i.e. networks with nodes having non-contiguous IP addresses) perfect tree is not guaranteed.

Ideally the number of messages sent over the network by a node is constant. It means that when the network is dense (e.g. there are no offline nodes, and there are no gaps in the IP addresses), the node communicates with its leader only. Hence, in contrast to full network scan, our election algorithm scales well for a large number of nodes (Section).

Level and offset are only useful to build subordination tree for linear topologies (all computers connected to a switch). For topologies where links between nodes are not homogeneous, it can be built by adding network latency into the mapping. Typically distributed system runs on top of *tree* physical topology (i.e. nodes connected to multiple switches), but geographically distributed clusters may form the topology with a few cycles. Measuring network latency helps a node to find a leader which is closest to it in physical topology. So, the algorithm for linear and non-linear topologies differs only in the mapping.

In the second phase of the algorithm – when a node has found its leader – a node measures network latency for subordinates of this leader and its own leader. So, only p nodes are probed by each node, but if the node changes its leader, the process repeats. It is difficult to estimate the total number of repetitions for a given topology as it depends on the IP addresses of the nodes, so there is no guarantee that the number will be smaller than for probing each node in the network.

The main goal of this algorithm is to minimise the number of packets transmitted over network per unit of time when finding the leader and the number of nodes is unknown, rather than maximising performance. Since changes in network topology are infrequent, the algorithm needs to be run only on initial cluster installation, and the resulting hierarchy of nodes can be saved to persistent storage for later retrieval in case of node restart.

So, for a distributed system subordination tree shape is close to that of physical topology, and for cluster with a switch the shape is determined by the maximal number of subordinates p a node can have. The goal of this tree is to optimise performance of cluster management tasks, which are discussed in Section V.

C. EVALUATION ON VIRTUAL NETWORK

Test platform consisted of a multi-processor node, and Linux network namespaces were used to consolidate virtual cluster of varying number of nodes on a physical node. Similar approach was used in a number of works [8]–[10]. The advantage of it is that the tests can be performed on a single machine, and there is no need to use physical cluster. Tests were repeated multiple times to reduce influences of processes running in background. Each subsequent test run was separated from previous one with a delay to give operating system time to release resources, cleanup files and flush buffers.

Performance test was designed to compare subordination tree build time for two cases. In the first case each node performed full network scan to determine online nodes, choose the leader, and then sent confirmation message to it. In the second case each node used IP mapping to determine the leader without full network scan, and then sent confirmation message to it. So, this test measured the effect of using IP mapping, and only one leader was chosen for all nodes.

Subordination tree test was designed to check that resulting trees for different maximal number of subordinate nodes are stable. For that purpose every change in a tree topology was recorded in a log file, and after 30 seconds every test run was forcibly stopped. The test was performed for 100-500 nodes. For this test additional physical nodes were used to accommodate large number of parallel processes (one node per 100 processes).

IV. RESULTS

Performance test showed that using IP mapping can speed up subordination tree build time by up to 200% for 40

nodes (Fig. 2), and this number increases with the number of nodes. This is expected behaviour since overhead of sending messages to each node is omitted, and predefined mapping is used to find the leader. So, our approach is more efficient than full scan of a network. The absolute time of this test is expected to increase when executed on real network, and thus performance may increase.

Subordination tree test showed that for each number of nodes stable state can be reached well before 30 seconds (Figure 3). The absolute time of this test may increase when executed on real network. Resulting subordination tree for 11 nodes is presented in Fig. 1.

V. DISCUSSION

Multiple levels of subordination are beneficial for a range of management tasks, especially for resource monitoring and usage accounting. Typical monitoring task involves probing each cluster node to determine its state (offline or online, needs service etc.). Usually probing is done from one node, and in case of a large cluster introducing intermediate nodes to collect data and send it to master helps distribute the load. In subordination tree each level of hierarchy adds another layer of intermediate nodes, so the data can be collected efficiently.

The same data collection (or distribution) pattern occurs when retrieving accounting data, and in distributed configuration systems, when configuration files need to be distributed across all cluster nodes. Subordination trees cover all these use cases.

VI. CONCLUSION

Preliminary tests showed that multiple level of subordination can be easily built for different number of nodes with fast IP mapping. The approach is more efficient than full scan of a network. Resulting subordination tree can optimise a range of typical cluster management tasks. Future work is to test how latency-based mapping works for geographically distributed systems.

ACKNOWLEDGEMENTS

The research was carried out using computational resources of Resource Centre "Computational Centre of Saint Petersburg State University" (T-EDGE96 HPC-0011828-001) within frameworks of grants of Russian Foundation for Basic Research (project no. 13-07-00747) and Saint Petersburg State University (projects no. 9.38.674.2013 and 0.37.155.2014).

REFERENCES

[1] G. Tel, *Introduction to distributed algorithms*. Cambridge University press, 2000.

[2] J. Brunekreef, J.-P. Katoen, R. Koymans, and S. Mauw, "Design and analysis of dynamic leader election protocols in broadcast networks," *Distributed Computing*, vol. 9, no. 4, pp. 157–171, 1996.

[3] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Stable leader election," in *Distributed Computing*. Springer, 2001, pp. 108–122.

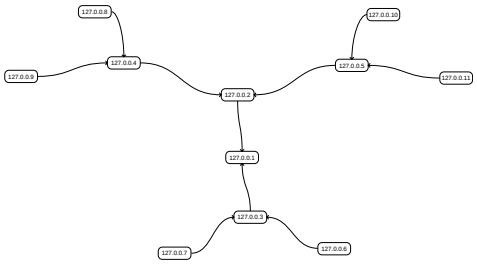

Figure 1. Resulting subordination tree for 11 nodes.

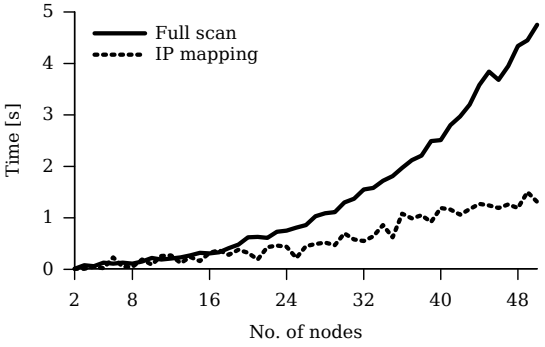

Figure 2. Time needed to build initial subordination tree with full scan of each IP address in the network and with IP mapping.

[4] P. Romano and F. Quaglia, "Design and evaluation of a parallel invocation protocol for transactional applications over the web," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 317–334, 2014.

[5] A. Degtyarev, "High performance computer technologies in shipbuilding," in *OPTIMISTIC — optimization in marine design, Mensch & Buch Verlag, Berlin*, L. Birk and S. Harries, Eds., 2003.

[6] I. Soshmina and A. Bogdanov, "Using GRID technologies for computations (in russian)," *Saint Petersburg State University Bulletin (Physics and Chemistry)*, vol. 3, pp. 130–137, 2007.

[7] S. Andrianov and A. Degtyarev, *Parallel and distributed computations (in Russian)*. Saint Petersburg State University, 2007.

[8] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 253–264.

[10] B. Heller, "Reproducible network research with high-fidelity emulation," Ph.D. dissertation, Stanford University, 2013.

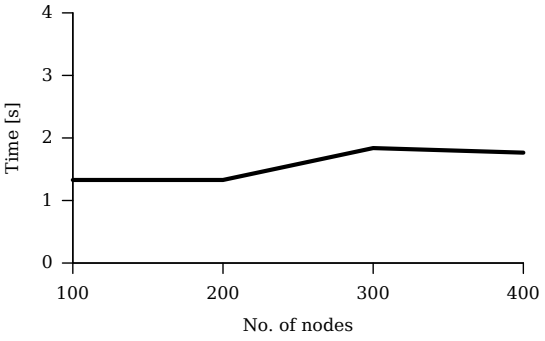

Figure 3. Time needed to reach stable subordination tree state for large number of nodes.