ISSN 2074-7888, Наукові праці Донецького національного технічного університету,
серія Проблеми моделювання та автоматизації проектування, 2011, Випуск 10 (197)

UDC 004.722.2, 004.057.4

Pavel Skvortsov (M. Sc.), **Ralph Lange** (Dr. rer. nat.),
Frank Dürr (Dr. rer. nat.)
Institute of Parallel and Distributed Systems, Universität Stuttgart, Germany
{pavel.skvorzov, frank.duerr}@ipvs.uni-stuttgart.de, mail@ralph-lange.de

OPTIMIZING MAINTENANCE COST OF P2P VORONOI OVERLAY NETWORKING

We present a P2P overlay network protocol based on Voronoi diagram and Delaunay triangulation. The proposed algorithms of node join and leave include the resolution of conflicts between distributed operations, by which concurrent networking is provided. The topology updating operations are performed having minimized maintenance cost measured in the number of messages. The proposed approach provides scalability by guaranteeing that for a large network the cost of a single join operation is constant. For proving this result, we evaluated the network maintenance cost by using an event-driven simulator.

Keywords: P2P network, Voronoi diagram, protocol, network topology.

Introduction

P2P overlay networks allow for scalable and robust data management of large scale distributed systems. This includes the following basic operations: 1) search for a data object with a given key (point query) or a key range (range query), 2) registration of the given data object with a node in the system with a key [1].

Depending on the organisation of internode links, the network overlay can be classified as unstructured or structured. The nodes in an unstructured overlay network are linked in arbitrary fashion, while structured network organisation is provided by an overlay pattern, which defines links for each node. In this work we consider a structured P2P overlay network, which is preferred because of the following disadvantages of unstructured networks: 1) they require higher message overhead for routing by using undirected search like flooding or random walk; 2) queries are not guaranteed to be resolved.

The existing structured P2P networks assume consistently hashed [2] keys and therefore they have the uniform distribution of keys by hashing. The consistent hashing scheme provides only local impact of the network topology changes caused by node joins and leaves. By such principle the cost of join and leave operations is optimized, in oppose to the traditional (non-consistent) hashing, where join/leave operations may cause the rebuilding of the whole network structure.

Today the widespread type of structured P2P network based on consistent hashing is the distributed hash table (DHT). DHT is a decentralised structured network, where each node is assigned to a unique interval or region of the keyspace. Examples of DHT implementations are CAN [3], Chord [4], Pastry [5], Tapestry [6], Tulip [7]. A non-DHT but a similarly structured overlay approach is HyperCuP [8].

DHT networks support point queries, but they have uniform distribution of keys due to hashing. The problem arises with locality-based range queries, which require non-uniform distribution of keys and at least 2-dimensional index space. A popular solution to these problems is the non-regular grid CAN [3], but this approach has the following drawbacks: 1) inherent problem with rectangular partitioning: a node without siblings (a neighbour node which has mirror cells) cannot be trivially removed from the network; 2) data load on node joins and leaves refers to only one other node, which leads to a skewed data distribution; 3) high variation of neighbours number, which may cause load misbalance as well as increase the cost of node joins and leaves.

To solve the described above problems, the Voronoi based partitioning of the key space has been proposed. Voronoi overlay (also known as Delanay overlay) is a well-known approach for location-aware networks, which is defined as follows. Given a set of nodes on the 2-dimensional plane, the Voronoi diagram is the subdivision of this plane into Voronoi cells, one for each node, so that the Voronoi cell for a node is the loci of all points closer to this node than to any other node. The links between nodes form the Delanay triangulation. Such location-aware distribution is suitable for a large network, where the size of nodes is negligibly small in comparison with the distances between them.

Several approaches for maintenance of Voronoi overlay have been proposed (for overview of the related work, see Section 3). The common drawbacks are: 1) node joins and leaves are not considered as possibly parallel events; 2) the networking maintenance is performed in a proactive fashion, which requires periodic exchange of neighbour tables (and therefore more overhead).

We propose a novel maintenance protocol for Voronoi overlay network, which creates and maintains the network topology based on Voronoi diagram and Delaunay triangulation. As distinct from the most other approaches, our protocol fulfils the following requirements:

1) Network management is performed in a distributed fashion, with no global supervision: a node can receive and send messages only to its own Voronoi neighbours.

2) Join/leave processes can occurring in parallel, and should run with minimal latency without causing inconsistency of the Voronoi network topology.

3) The protocol minimises the maintenance cost measured as the number of messages to be transmitted between the nodes of the overlay network for its maintenance.

The remainder of the paper is structured as follows. In Section 2 we discuss related work; then we describe our system model in Section 3. In Section 4 we present the basic protocols and algorithms for a peer joining or leaving the Voronoi network and analyse their communication cost. Then we discuss concurrent join or leaves in Section 5. In Section 6 we evaluate the cost of our protocols. Finally, the paper is concluded in Section 7.

Related Work

There are various approaches for Voronoi/Delaunay based networks. Liebherr et al [9] describe the Delaunay networking mechanisms with full networking functionality, where both network building and maintenance are provided by periodic messages between the nodes. The approach is very reliable, but causes excessive communication overhead.

The incremental construction in the work of Hu & Liao [10] requires additional knowledge about non-geographic neighbours, since all the nodes within AOI (Area of Interest) of a node are considered as its neighbours. Various actions (move, jump, chat, trade) of nodes are realised.

GeoPeer of Ujo & Rodrigues [11] uses LRC (Long Range Contact) mechanism for routing. To create and maintain the Delaunay topology, nodes are periodically exchanging messages with their geographic neighbours, which leads to an excessive communication overhead.

Kang et al [12] describes routing and join algorithms for Delaunay networks, where the join algorithm is performed through distributed message processing. The node join events are supposed to occur sequentially; no conflicts between distributed messages are considered, as well as leaves of nodes.

Ohnishi et al [13] propose an incremental algorithm for building the Delaunay overlay network. The algorithm assumes that nodes exchange all the information about their neighbours as well as so-called non-neighbours, until the neighbour lists of all nodes are complete. This principle also maintains network having parallel networking operations.

Beaumont et al. [14] Long Range Contact (LRC) mechanism, as well as the additional set of so called “close neighbours” for each node. Some problems of routing and maintenance are analysed, but the disadvantage is that parallel joins are not considered in their work.

Lee and Lam in [15] and [16] proposed distributed join and leave algorithms for Delaunay based network, including the mathematical proof of the algorithms’ correctness. The fixing of possible inconsistency caused by concurrent joins is separated from the algorithm itself and is being performed only by excessive heartbeat messages.

System Model

Our target system consists of a multitude of nodes, where every node has the following information:

1) Unique address (ID), which is used for communication between nodes;

2) Position in terms of 2D logical coordinates x and y, to define the node's placement in the network service area;

3) The boot node – namely, the address and position of a node (we assume that this node is randomly selected) which is already integrated into the existing network topology. Knowing the boot node, a new node can start the process of joining the network.

Based on the geographic position of a node, our protocol assigns a separate Voronoi cell for this node by determining its Delaunay neighbours. After a successful join, the new node knows the following:

1) Voronoi neighbours – the neighbour table which includes addresses and positions.

2) Logical coordinates of data items (users and any other objects, which are dependent to the node, being in its area of responsibility).

All nodes of the network are connected through the Internet-like infrastructure. Here, Delaunay triangulation represents the set of links between nodes in the Voronoi network. Nodes communicate with each other by messages transmission. In our work we consider only messages for network maintenance used by the Voronoi protocol, and do not consider messages for transferring user data between nodes.

The proposed overlay protocol provides the consistency of network's Voronoi topology by implementing the maintenance algorithms. On the lower level, we assume that nodes exchange messages using a reliable TCP-like transport protocol. Non-delivered messages are being resent; duplicate messages rejection is supported by unique identification of every message, knowing the address of its sender and the sender's local ID of that message.

Basic Network Operations

In this section we describe the basics of approaches for regular (re)building of Voronoi network, which include algorithms of node join and leave. Here we do not consider that the presented network operations may occur in parallel for the neighbouring nodes; the protocol's improvement for concurrent operations follows in the next section.

Node Join

In general, the join algorithm consists of two phases. First, to route until the destination node A which can be a vertex node of the new node's

circumscribing triangle *ABC*, or a vertex of the nearest convex hull segment *AB*. The second phase is to determine the full list of the new node's neighbours and create a new Voronoi cell inside the existing diagram, or to extend the existing diagram's convex hull with the new node (if it has joined outside).

We assume that the new node *N* knows at least one node *F* (the boot node) which already belongs to the Voronoi network. The routing starts from *F* to find the region containing geographic Voronoi neighbours of *N*. During the routing process the current node must choose the next one on every step, knowing only its neighbours. The choice of the next node is done by the routing algorithm similar to the described in [17]. The main algorithm's goal is to find the circumscribing Delaunay triangle for *N* (or the nearest segment, if *N* is outside the actual Delaunay triangulation). Thus, the routing relies on Delaunay links and not on the neighbour's distance to the target point.

If a node *S* with neighbours $N_1...N_k$ must forward a message to the destination node *A*, it performs the following steps during the routing:

1) Check whether *S* is actually *A*, or if *S* has *A* among its neighbours. If yes, destination node *A* is found, otherwise go to step 2.

2) Find the intersection between imaginary *SA* line and N_iN_{i+1} Delaunay segment.

3) Check, which node of the N_iN_{i+1} segment is closer to *A*.

4) Send message to this neighbour to set it as *S*; go to step 1.

The routing process arrives at the node *A*, having detected one of the following 3 situations:

a) There is a Delaunay triangle *ABC* which circumscribes *N*. It means that *N* is located inside the actual Delaunay triangulation and can join network through regular (internal) algorithm (see Section 4.2.1).

b) *N* is outside of any existing Delaunay triangle, and the nearest segment *AB* of convex hull was found. Thus, the algorithm of an external join will expand the convex hull of the Delaunay triangulation with *N* (see Section 4.2.2).

c) Previously there were only two or less nodes in the network (no further processing is required).

In the first two situations, the complexity of the required messages number, to which the join algorithm approximates, is:

$$m_{join} = r + (2n - 1), \tag{1}$$

where *r* is the number of messages which are sent to route from *F* to *A*; *n* is the neighbours number of *N* after join is complete.

The evidence for (1) is the following: it is assumed that every neighbour (except *A*, which actually initiates the join algorithm – this explains "minus 1" in the formula) receives one message to add *N* to its neighbour list, and sends one message for *N* to add this node to its neighbour list.

Internal Join

The internal join algorithm for a new node N is used if the circumscribing triangle for N is found – this means that N does not expand the Delaunay triangulation but joins in the internal area of it. The starting point for the algorithm is node A of the circumscribing triangle ABC, at which the routing from the boot node was finished. A looks among its neighbours for the nodes which now must be the neighbour of N, and it sends in parallel a "construct" message to B and C for them to do the same. The next potential neighbour of N, e.g., from the sight of node A, is the B_i, where AB_i is the next (clockwise from AB) Delaunay segment with minimal angle BAB_i (see Fig 4.1). If B_2 satisfies the determinant condition [18] (i.e., B_i is the Voronoi neighbour of N), the algorithm continues for both AB_2 and B_2B segments, and so on.

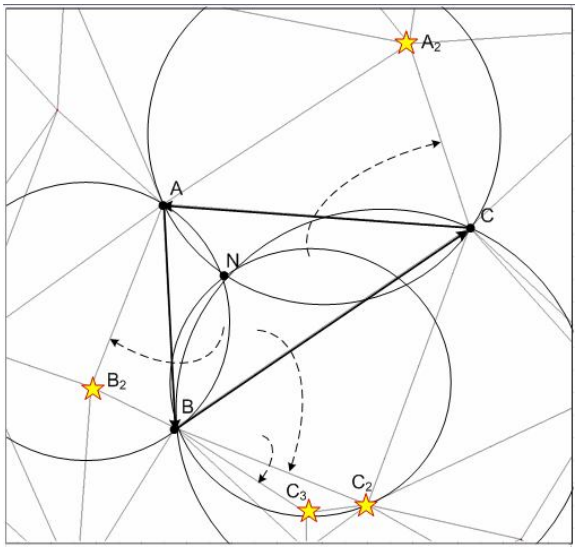

Figure 4.1 – The internal join algorithm

Each new found neighbour adds N to its neighbour list and independently sends the "update" message to N. Thus, in the end the new joined node N has its own complete neighbour list.

External Join

The external join algorithm is used when the new node N joins the external area of network's Delaunay triangulation, and therefore the circumscribing triangle for N does not exist. In that case only the nearest convex hull segment is found through the routing from boot node.

The starting point is the convex hull node A of the found segment AB. The message "find convex hull neighbours" is being forwarded in both A and B directions through convex hull, if among neighbours of the current node there are convex hull nodes, for which exists a non-intersected Delaunay connection

with *N*. Every found convex hull neighbour of *N* sends the "construct" messages in the same way as it is described in Section 4.2.1. The "update" messages are being sent to *N* in the similar principle as well.

If after the join process a previously external node became internal (is not a part of convex hull anymore), it sends the message about its convex hull status change to all its neighbours. The convex hull status of every neighbour must be represented in the neighbour list, since it is relevant for routing.

Node Leave

The node leave is distinguished from the node failure. The leave is about regular and controlled move out of the network of a single node, initiated by its own request. If node *L* of a regular Voronoi network initiates its leave process, it sends the message "delete me" to all its neighbours. With this message nodes receive the neighbour lists to the rest *L*'s neighbours. Having got this information, each neighbour can update its neighbour list independently, using the determinant condition [18]. Therefore, the number of messages required for the leave process is: $m_{leave} = n$ (n is the neighbours number of L).

Concurrent Network Management Protocol: Joins and Leaves

Here we explain how the concurrent management of network is supported. We show how conflicts between neighbouring join operations may occur, and how we can fix the resulting topology inconsistency using the *Credit method*.

Conflicts between Parallel Operations

The previous section's algorithms assumed that nodes join and leave the system sequentially. Here we introduce methods to deal with concurrent operations. Parallel joins and leaves are desirable because this minimizes the total latency. Some separate network's regions might be reorganised in parallel, if they do not affect each other during this process. The problem is that usually during the update process some node has incomplete list of neighbours or another kind of incorrect information, and some another node can send to it a request to get this information for another process. This situation is unwanted and critical, but cannot be avoided in distributed networking.

A possible solution is to have an external control to prevent operation to take place before the previous process ends. However, this approach: a) requires a global supervising mechanisms, which is contrary to the distributed networking principles; b) having high churn rate, it requires queues of the joining nodes, which may lead to monotonously increasing delays of join requests. Thus, in a large and dynamic network, if time intervals between requests are shorter than the runtime of the algorithm, the node join algorithm

must also provide the recovery of temporal inconsistency caused by parallel joins.

In terms of a join operation, between the start and the end of the process, every node participated in this process can be in one of the following 3 states:

1) *Anticipant* state: yet no changes regarding the actual join process are done (at this time process works still with other nodes, and the "construct" message to this node is yet to be sent);

2) *Transitive* state: some requests are processed at this node, and some are not (or not received yet);

3) *Accomplished* state: join process is completed and the Voronoi diagram around the new node is regular.

While node is in *transitive* state, its neighbour list is inconsistent. Actually in the *anticipant* state of a node its neighbour list is already non-relevant: the new node already exists, but is not integrated into network. If another node begins its join process, the message initiated by it can request information stored by a node which is already in *transitive* state. If it requests information from a node in the *anticipant* state, this leads to the opposite situation. The result is an incorrect neighbour list for both of the new nodes.

In a large Voronoi network, having low rate of joins, the majority of joins may proceed without preventing each other because they are localised. However, the described conflicts must be detected and solved in order to allow the network run in parallel mode and have a consistent network topology. First of all, to prevent the forwarding of incorrect information (about node's neighbours), the start and the end of *transitive* state for every involved node must be determined.

Determining Join Completion with Credit Method

A node which has initiated the join process is set to be responsible for that, by assigning and releasing the *busy* flag for all the participant nodes. The main idea of the *Credit method* is to determine when the join process is complete through collecting all *credits* distributed during the join process. When all credits are collected, we can release *busy* flag of all participant nodes at the same time.

Typically the *transitive* period starts with the receiving of "construct" message. To mark the end of *transitive* state, the initiator node (say, *N*) sends the "end" message to all its neighbours, which it got when the join process has finished. Since every single transaction is independent and has no view of the whole process, the problem here is to determine, when the joining is finished. The question "Has *N* received "update" messages from all its neighbours or not?" is a problem of distributed computing. To solve it, *Credit method* is used [19].

Each node which is currently involved as a new neighbour of *N* receives its *credit* value with "construct" message. In the beginning every vertex node of the circumscribing triangle gets *credit* = 1. If a vertex node (say, *A*) finds a new neighbour for *N* amongst neighbours of *A* (say, A_1), A divides its credit between itself and A_1. Thus, A_1 receives its *credit* = 0.5 with "construct" message from *A*. This algorithm runs before the next found node has no neighbours to extend the Voronoi cell of *N*. As a result, the *credits* sum of all "successors" of each one of 3 original vertex nodes remains 1, therefore the total sum is always 3.

For the external case of join algorithm the start *credit* values are 1.5 for every node of the nearest segment. This *credit* is also divided if a node finds another neighbour for *N*; therefore the total sum of all *credits* for every separate join process (as well as at any moment of time between start and end for both internal and external algorithms) remains 3. With every "update" message to *N* from its new neighbour, *N* receives the *credit* value of sender and adds it to the *credits counter*. When this counter achieves 3, it means that the join process for *N* is complete and *busy* flags of all neighbours now can be released. Thus, *N* sends the broadcast message "end" to all its neighbours and removes its own *busy* flag as well. Only after releasing its *busy* flag a node is ready again to receive and process requests; before that the receiving of a request for another initiator's process indicates a conflict between processes.

Conflict Resolution

There are two types of the concurrent conflicts: 1) process/routing conflict, which can be solved by delaying the join of only one of two conflicting nodes; 2) process/process conflict, which requires the rollback of both operations. The situation with concurrent leaves is considered separately. For the conflict resolution we use distributed algorithms, which rely on the node states defined by *Credit method*.

Resolving Process/Routing Conflict

This is the situation when the join process for a node, say N_1, blocks the routing stage of the join process (see Section 4.1) for node N_2. In that case the conflict occurs when the neighbour list of N_2 is empty, as well as no changes to the network topology were initiated by N_2.

Thus, the *busy* node (node in *transitive* state) which has therefore detected the conflict (and which cannot forward the routing because of inconsistency of its neighbour list) sends the "rejoin" message to N_2. Since N_2 has not initiated any network change yet, other nodes do not have to be informed (no rollback for them is required). N_2 will try to join the network after a certain timeout, when N_1 is supposed to finish its join process and to set free the *busy* node on the path of the routing, which was stopped during the previous attempt.

Resolving Process/Process Conflict

This type of join conflict may occur when the parallel joining nodes have already found their circumscribing Delaunay triangles and independently began to change the actual Voronoi network topology. It can lead to the topological inconsistency, if the joining new nodes either are mutual neighbours (i.e. should be neighbours after the correct join processes) or will have common neighbours.

In Fig. 5.1 nodes N_1 and N_2 are beginning join processes. Nodes F and G should be their common neighbours, and N_1 and N_2 should be considered as mutual neighbours. It means that without conflict resolution, these two parallel processes may ignore each other, use incomplete neighbour lists and therefore do not establish all links necessary for consistency the topology.

The basic idea of conflict resolution is after detecting concurrent joins, to rollback both join processes and re-start them after different timeouts.

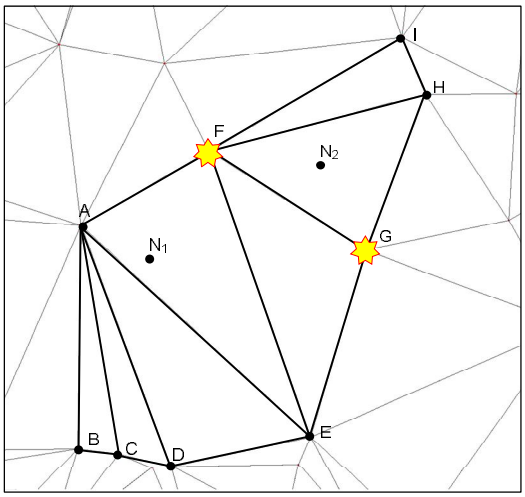

Figure 5.1 – Process/process conflict emerging

If a *busy* node receives messages from another join process, it defines it as conflict and takes action. The node which detected a conflict (F or G), knowing the addresses of both new nodes, sends them the message “rollback and rejoin”. The new nodes send to all the currently found neighbours a message to restore their last valid neighbour lists (which is being stored with every *busy* assigning, before the new join begins the update of list). Then, after the randomly chosen timeouts with enough large variation, both new nodes try to join the network again (supposed to do it at different points in time).

Note, that we cannot stop only one join and continue another, because both processes might already have used the incorrect information of nodes in *anticipant* state.

If more than 2 nodes are conflicting, the resolution does not differ from the two nodes conflict in terms of algorithm. Such a situation only extends the conflict domain and increases the possibility of rejoin conflict.

Dealing with Concurrent Leaves

Leave processes are being executed in terms of only one time step – single broadcast of the message "deconstruct neighbour" – and do not mark the involved nodes as *busy*. It means that we cannot detect the concurrent neighbouring leaves by using the mechanism of conflicts detection, described in the previous Section 5.3. At the same time, the problem remains, because if a leaving node sends a message about this to its neighbour which just left the network concurrently, the nodes around them do not have the full list of nodes to rebuild the local Delaunay diagram.

Here we assume that if a node leaves the network, it does this not immediately but still can receive and send messages for a time sufficient for the possible delayed reaction of other nodes. We say that node in such situation is in *suspended* state: when it has already left the network topology but can react to delayed messages from its former neighbours (or even some other messages from non-neighbours).

If a *suspended* node receives the "deconstruct neighbour" message from its former neighbour, it repeats the leaving broadcast of "deconstruct neighbour" messages, with the extended node list sent with each message:

1) *A suspended* node A knows its former neighbours $A_1 \ldots A_n$.

2) A receives from B the "deconstruct neighbour" message with list $B_1 \ldots B_k$.

3) A sends the "deconstruct neighbour" message to $A_1 \ldots A_n$ and $B_1 \ldots B_k$, which contains both all $A_1 \ldots A_n$ and $B_1 \ldots B_k$ nodes.

In case when the receiver node of a "deconstruct neighbour" message is *busy* (because of a parallel join process), is receives this message but does not execute the corresponding rebuilding of its neighbour table only after its *busy* status is changed to *active*.

Evaluation

Basic Cost for Building the Overlay Network

To evaluate the basic message overhead for network management, we only consider sequential joins: each new node joins only after the previous join process is fully completed. Therefore, we can determine the minimum maintenance cost, which take place independently from any conflicts.

Here we distinguish 3 types of the networking cost: main (needed for new a Voronoi cell construction), hull, and routing messages. The resultant numbers of these messages used for network construction with different sizes are shown in Fig. 7.1, as well as the number a node's neighbours in the resulting network. All values here are average number of messages per node. Our results confirm that the average number of a node's neighbours in the Voronoi network

approximates to 6 [20] if we have uniform distribution and enough many joined nodes. “Enough many” means 100-200 joined nodes, as we can see in the diagram in Fig. 7.1.

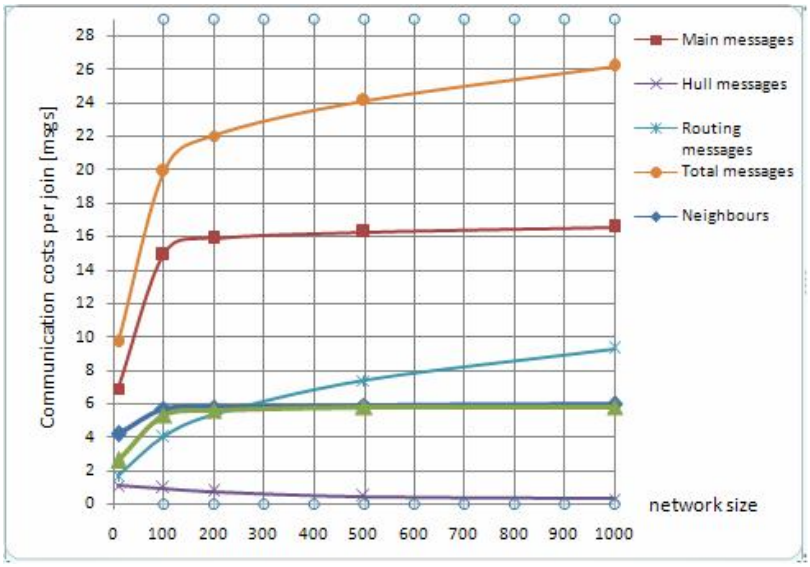

Figure 7.1 – Basic communication cost in sequential mode

We can see that for a large network the cost of a single join is predictable getting closer to a constant value, if the network size is stable. For a large network with a varying size the only non-constant component of the node join cost structure is the messages number of the routing for join.

Cost of the Concurrent Networking

Here we evaluate the network construction in terms of concurrent joins, for what the parallel mode of simulation is used. In oppose to the sequential mode evaluated in Section 7.1, it means, that each new join may occur before the previous join process is complete. The number of new node joins per 1 second is the join rate; it’s defined once for every simulation of network construction.

Different network sizes and join rates provide various fractions of messages for routing and non-routing (management and recovery) messages in the total number of transmitted messages (Fig. 7.2). The reason for this is that more conflicts occur in the beginning of the network construction (because of network’s small size) and those nodes rejoin after the relatively long timeouts. Since the network gets larger, nodes joins cause fewer conflicts and the average value of non-routing messages number stabilises, while the number of routing messages increases. Thus, the upsurges of the curves given in the diagram of total cost in Fig. 7.2 are caused by the numerous conflicts and recovery (non-routing) messages, while the constant rising is determined by the routing

messages increase. For larger networks the average cost are getting lower because of lesser number of conflicts and rollbacks.

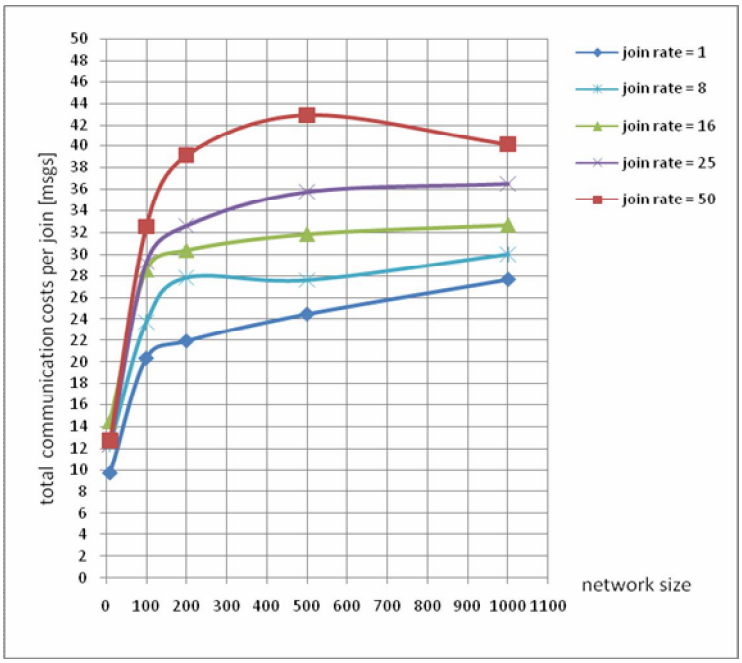

Figure 7.2 – Parallel simulation results for various join rates

For join rates 1 and 4 the results are very similar as for the sequential mode because of relatively long delays between joins. Rates 8 and 12 significantly increase the average values for small networks by using a lot of algorithmic messages in the beginning of building process. After stabilisation the parameter's increase is achieved mostly by routing messages. The much higher rates 25 and 50, however, show another trend: they causes so many rollbacks (as in the beginning, so later after conflicts between rejoins) that the rejoin events become distributed uniformly in time, as well as average messages number.

Conclusion

First we motivated the application of Voronoi diagram as network topology for its balanced data distribution between nodes. Then, we proposed a new network protocol for managing a Voronoi overlay network. The main advantage of our approach is that it supports concurrent join and leave operation without using heartbeat messages. Our evaluation showed that the average cost of networking operations is limited by a constant value. By excluding the routing messages, the construction of a new Voronoi cell is localised and its cost does not growth together with the network size.

References

1. J. Gao Efficient support for range queries in DHT-based systems: Technical Report CMU-CS-03-215 / J. Gao and P. Steenkiste. – Carnegie Mellon University, 2003.

2. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web / D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, and R. Panigrahy // In Proceedings of the 29th Annual ACM Symposium on Theory of Computing. – May 1997. – P. 654-663.

3. A Scalable Content-Addressable Network / S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker / In ICSI Technical Report. – 2001, Jan.

4. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications" / I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan // ACM SIGCOMM. – 2001.

5. Rowstron "A. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems" / A. Rowstron and P. Druschel // IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany. – 2001, Nov. – P. 329-350.

6. Tapestry: a resilient global-scale overlay for service deployment / B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J. Kubiatowicz // IEEE Journal on Selected Areas in Communications. – 2004. – P. 41-53.

7. Practical locality-awareness for large scale information sharing / I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron // In Proceedings of IPTPS. – 2005.

8. A scalable and ontology-based P2P infrastructure for semantic web services / M. Schlosser, M. Sintek, S. Decker, W. Nejdl // In Peer-to-Peer Computing. – 2002, 104-111.

9. Liebeherr J. Application-Layer Multicasting with Delaunay Triangulation Overlays / J. Liebeherr, M. Nahas and W. Si // IEEE J. Selected Areas Comm. – 2002. – V.20. – P. 1472-1488.

10. Hu S.Y. "Scalable Peer-to-Peer Networked Virtual Environment" / S.Y. Hu and G.M. Liao // Proc. ACM SIGCOMM. – Wksp. on NetGames, Aug. 2004.

11. Araujo F. "Geopeer: A location-aware peer-to-peer system" / F. Araujo and L. Rodrigues // Department of Informatics, University of Lisbon, DI/FCUL TR 03–31. – 2003, December.

12. P2P spatial query processing by Delaunay triangulation / Bog-Ja Lim, Hye-Young Kang, and Ki-Joune LiIn // Proceedings of Workshop on Web and Wireless GIS. – 2004. – P. 136–150.

13. Masaaki Ohnishi "Incremental Construction of Delaunay Overlaid Network for Virtual Collaborative Space" / Masaaki Ohnishi, Ryo Nishide, Shinichi UeshimaIn // Proc. of Conference on Creating, Connecting and Collaborating through Computing (C5'05), IEEE CS. – Kyoto: Japan, January 2005.

14. VoroNet: A scalable object network based on voronoi tessellations / Beaumont O., Kermarrec A.-M., Marchal L., Rivière É. // In proceedings of IPDPS 2007. – 2007, March.

15. Dong-Young Lee Protocol Design for Dynamic Delaunay Triangulation. ICDCS 2007: 26 / Dong-Young Lee, Simon S. Lam.

16. Dong-Young Lee "Efficient and Accurate Delaunay Triangulation Protocols under Churn" / Dong-Young Lee and Simon S. Lam // The University of Texas at Austin, Department of Computer Sciences. Report# TR-07-59 (technical report). – November 9, 2007. 14 p.

17. Prosenjit Bose Online Routing in Triangulations.ISAAC'99:113-122 / Prosenjit Bose, Pat Morin.

18. Sugihara K. Construction of the Voronoi Diagram for One Million Generators in Single-Precision Arithmetic / K. Sugihara and M. Iri // In Proceedings of IEEE, 80(9):1471-1484, 1992.

19. Mattern F. Verteilte Basisalgorithmen / F. Mattern. – Springer-Verlag, 1989.

20. Computational Geometry: Algorithms and Applications (Third Edition) / Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. – Springer-Verlag, 2008. – 386 p.

Надійшла до редакції 12.10.2011
Рецензент: д-р техн. наук, проф. Святный В.А.

П.В. Скворцов, Р. Ланге, Ф. Дюрр
Штутгартський університет, Німеччина

Оптимізація обслуговування P2P оверлейної мережі Вороного. В цій роботі ми представляємо протокол для оверлейної піринговой мережі, заснований на діаграмі Вороного і тріангуляції Делоне. Запропоновані алгоритми приєднання і від'єднання мережевого вузла включають в себе вирішення конфліктів між розподіленими операціями, за допомогою чого забезпечується функціонування операцій зі зміни мережевої топології в синхронному режимі. Операції відновлення топології виконуються з мінімізацією витрат, які ми вимірюємо в кількості повідомлень. Пропонований підхід забезпечує масштабованість: гарантується, що у великій мережі вартість окремого приєднання вузла наближається до постійних значень. За допомогою емулятора оцінено вартість операцій по зміні мережевої топології.
Ключові слова: P2P-мережі, діаграма Вороного, протокол, мережева топологія.

П.В. Скворцов, Р. Ланге, Ф. Дюрр
Штутгартский университет, Германия

Оптимизация обслуживания P2P оверлейной сети Вороного. В этой работе мы представляем протокол для оверлейной пиринговой сети, основанный на диаграмме Вороного и триангуляции Делоне. Предложенные алгоритмы присоединения и отсоединения сетевого узла включают в себя разрешение конфликтов между распределенными операциями, с помощью чего обеспечивается функционирование операций по изменению сетевой топологии в синхронном режиме. Операции по восстановлению топологии выполняются с минимизацией затрат, которые мы измеряем в количестве сообщений. Предлагаемый подход обеспечивает масштабируемость: гарантируется, что в большой сети стоимость отдельного присоединения узла приближается к постоянному значению. С помощью программы-эмулятора была оценена стоимость операций по изменению сетевой топологии.
Ключевые слова: P2P-сети, диаграмма Вороного, протокол, сетевая топология.