Resource sharing architecture for cooperative heterogeneous P2P overlays

Citation for published version (APA):

Exarchakos, G., & Antonopoulos, N. (2010). Resource sharing architecture for cooperative heterogeneous P2P overlays. *Journal of Network and Systems Management*, 15(3), 311-334. <https://doi.org/10.1007/s10922-007-9069-6>

DOI:

[10.1007/s10922-007-9069-6](https://doi.org/10.1007/s10922-007-9069-6)

Document status and date:

Published: 01/01/2010

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Resource Sharing Architecture For Cooperative Heterogeneous P2P Overlays

Georgios Exarchakos · Nick Antonopoulos

Published online: 21 June 2007
© Springer Science+Business Media, LLC 2007

Abstract Resource requirements and availability in heterogeneous networks may frequently vary over their lifetime; thus producing equally variant overloaded and under-loaded situations. Typical architectures cannot cope with the frequent availability fluctuation of reusable, non-replicable and highly dynamic resources (such as network capacity). This paper proposes an unstructured P2P overlay for sharing resources between underutilized and overloaded networks. Its aim is to satisfy the excessive resource demands of some networks by using free resources from others given the high failure rate and unstable availability of these resources in wide networks. We describe and analyze the proposed Capacity Sharing Overlay Architecture and show, with extensive simulations, its ability to provide remote underutilized capacity to underlying networks, even in the presence of high node failure rates, helping the networks to handle more user queries.

Keywords Peer-to-peer · Resource sharing · Heterogeneity · Reactive systems

Introduction

Heterogeneous distributed applications deployed on different networks may have quite variable network throughput performance requirements during their lifetime. We define the Network Capacity as the number of user queries a node can process within a time unit. The Network Capacity depends on the combined communication and computation throughput of that node. In case the application's workload overcomes the available network capacity (overloaded situation), new nodes are

G. Exarchakos (✉)
Computing Department, School of Electronics and Physical Sciences,
University of Surrey, Guildford, Surrey GU2 7XH, UK
e-mail: g.exarchakos@surrey.ac.uk

N. Antonopoulos
e-mail: n.antonopoulos@surrey.ac.uk

required to join the network to serve the demand. On the contrary, when the application produces traffic that fewer nodes could efficiently serve (underloaded situation) the network application may free some of them and increase the remaining nodes' utilization.

In this research, we try to address the problem of network workload fluctuations by changing appropriately the network size. While underloaded networks may provide underutilized nodes, overloaded ones may use them to serve their excessive traffic. Moving capacity from one network to another may improve the utilization of spare capacity and the reliability of networks even in high workload situations. Node and capacity sharing will be used interchangeably this point onwards since network capacity is closely related to the size of a network. In our architecture two steps are necessary to share capacity between networks: discovery, move of capacity. While the first step locates the appropriate nodes the second one fetches them to the requestor network.

CSO, Capacity Sharing Overlay, is a P2P management system that enables the sharing of reusable resources and specifically network capacity. Network capacity is a non-replicable, reusable, stochastically available resource and only one instance of it may exist within a network and no more than one user may use it each time [1]. This architecture facilitates the cooperation of heterogeneous networks for improving the utilization of the spare (currently not used) capacity in the whole system. The overlay consists of a set of interconnected servers each of which represents the nodes of an underlying network.

The main objective of the CSO is to provide a set of services to the interconnected networks to enable the node sharing. These services are the join/leave and query/response processes that can be used by any server. A server participates in the overlay if it knows one or more servers that already have joined the CSO. Therefore, the join process informs the new server about the existence of neighbours that will make the former accessible by the other servers and, vice versa, resources of the other networks accessible from the new server. Any server may leave the overlay at any time without notifying any of its neighbours. The query/response services define the behaviour of each server while it is active within the overlay.

Related Work

Any centralized approach of interconnecting all the underlying networks' servers would suffer from the high workload from frequent queries and advertisements of the requested and available capacity respectively. The high rates of leave/join actions of the servers and nodes would cause extra significant update overhead. In case of a failure of the central manager, no capacity sharing would be possible.

Adopting a more distributed approach using P2P Networks solves the single-point-of-failure and reliability problems of the centralised one. Replication of resources [2, 3] may increase the throughput performance of the overlay network since it increases the availability of the same resource [4]. Advertisement [5] or gossiping may direct the query faster to the resource provider thus reducing the latency. DHT-based P2P systems such as CAN [6], Chord [7], Pastry [8] and

Tapestry [9] can guarantee successful discovery if the requested resource is available in the network within $O(\log n)$ messages [10]. However, replication or gossiping/advertisement as well as informed resource discovery techniques of P2P Networks are not applicable to the discovery of network capacity since it is a non-replicable reusable with high fluctuations on its availability resource. Organizing the overlay servers in a Structured P2P would require the use of its lookup function each time a node joins the system resulting in a high maintenance cost.

Existing research on high throughput computing has produced several solutions to the issue of reusable resources discovery especially storage capacity and CPU cycles. Condor [11, 12] is one of the most mature high throughput computing technologies. It is a distributed job scheduler providing the appropriate services for submission and execution of jobs on remote idle resources even if they are under different administrative domains. A central manager receives the advertisements of the available resources and tries to submit the queued jobs to the appropriate ones, which report back to the manager the execution state of each job. The central manager along with the idle resources constitutes the condor pool. The flocking [13, 14] was introduced to statically link several condor pools and share resources between them. Manual configuration of a pool's neighbors is required thus limiting the adaptivity of the system in case of dynamic changes in resource availability. It is also assumed that the pool managers run on reliable machines [15] since their failure can prevent the execution of new jobs. These problems can be approached with a Pastry [8] self-organizing overlay of condor pools [15]. The condor pools are organized into a ring and proactively advertise their idle resources to their neighbors so that they can choose these advertised resources whenever necessary. Unfortunately, this P2P-based flock of Condors requires a substantial maintenance overhead for updating the proximity-aware routing tables since it is based on the advertisements of available resources. If the availability of the resources changes very frequently these updates need to be frequent and therefore high maintenance costs.

Finally, an important feature of Condor Flocking is that the execution machines are always managed by the same managers. Thus, every new discovery of the same/similar remote resources by the same manager follows the same procedure. Given that the required network capacity could frequently exceed the locally available, the local manager would forward equally frequent queries seeking almost the same amount of capacity; thus resulting in a significant number of messages.

The benefits of P2P overlays [10, 16] for the discovery of reusable resources have been identified and used in P-Grid. P-Grid, identifying the update overhead posed by available resources' advertisements on DHT-based overlays, uses a tree-based distributed storage system of the requesting resources' advertisements [17]. The resource providers locate in this tree the requestors they can serve and offer themselves for use. While other structured P2P networks hash the indexing keys, thus limiting the searching capabilities, P-Grid enables complex queries. The organization of this overlay raises a number of concerns about its scalability in case of large highly dynamic networks since an update action of one advertisement could propagate to many peers.

Sensor Networks is another field that uses the benefits of P2P Networks to achieve reliable cooperation of networked sensors. Recent research on P2P-based

end-to-end bandwidth allocation [18] proposes a wireless unstructured overlay of sensors. Initially a central peer possesses all the bandwidth and distributes it on-demand and every query is broadcasted to all peers. This system cannot be applied to the case of network capacity sharing since it makes the assumption that the available bandwidth within the whole network is known *a-priori* and that the topology of the network remains the same. Finally, it is suitable for wireless environments where the cost of broadcasting is the same to unicasting.

All the systems described above are efficient in the context they were developed for but they are insufficient in the context of network capacity. Network characteristics may change extremely fast so that any advertisement and/or indexing scheme could result in frequent updates with a high cost on messages. G-ROME [19] builds an unstructured overlay on top of ROME-enabled Chord [7] ring networks and its aim is to discover underutilized nodes in ROME [20] node pools and transfer them to requestor pools. Trying to address the problem of the frequently changing capacity availability, G-ROME uses an unstructured overlay of ROME bootstrap servers that represented Chord idle resources. The proposed architecture (CSOA) goes one-step forward decoupling G-ROME from Chord rings and supporting a more heterogeneous environment of interconnected networks.

Design Assumptions

Fault-tolerance and low maintenance costs are very important in such dynamic environments since they may affect the scalability of the overlay. Applicability and full support of network heterogeneity are two more requirements approached by creating loosely coupled servers and minimal service clients. That is, any member of the underlying network can use the server's capacity sharing services, no matter what the structure, size and discovery mechanism of the network is. The workload fluctuations and underlying networks' heterogeneity may produce queries with different requirements and generate capacity availability of any size.

It is assumed that the moving nodes can accept incoming connections allowing the remote submission or configuration of the destination network's application client to the node, thus making the concept feasible. The requested capacity will be served by a single server only and no reservation scheme is deployed to enable partial answers. Avoiding partial answers simplifies the discovery mechanism and helps the overlay to deal with the high failure rates of nodes and servers and variable availability of reusable resources in P2P networks. Reserving unreliable nodes could easily result to deadlock situations and increase the query failure rates.

Capacity Sharing Overlay Architecture (CSOA)

The proposed architecture builds an unstructured P2P overlay that interconnects several servers. Each server (CSOS) represents an underlying network: a mobile device, workstation, cluster, supercomputer or any arbitrary interconnection of

query is satisfied if the capacity availability in the Node Pool is lower than this safety level.

- *Query Processor (QP)* processes any incoming query and realises the communication protocol of the CSOA. It caches any internal and external query for a given period of time. It checks the Node Pool whether the query can be satisfied and if so it responds back or forwards the query. In case of internal queries, it waits for the answers. As soon as an answer is received it merges the capacity that the answer contains with the capacity the local Node Pool has reserved, if any, and responds back to the originator node.
- *Underlying Network Relocator (UNR)* resides in the underlying network nodes and is responsible for monitoring the network application and controlling the node movement from one network to another. It is used for deciding whether to register with or query a server or is invoked by the remote requestor node to move itself to the latter's network.

Overlay Interaction Messages

The communication protocol of CSOA defines five messages for registering an underloaded node with a server, asking the overlay for under-loaded nodes, responding with a list of nodes and realizing the actual move of a node from one underlying network to another. Table 1 gives the list of messages that are involved in the interactions between CSOA entities.

Figure 2 demonstrates the interaction of nodes and servers in case of server and node registration, query forwarding and node movement.

Each CSOA message has a header and a payload. The header has an invariable size and consists of four fields and the payload has variable number of fields and size, shown in Table 2.

1. Message ID: a 16-bit message unique identifier
2. TTL: maximum number of steps the message can travel within the overlay
3. Type: a code specifying the type of the message's payload (register: 0×01 , query: 0×10 , response: 0×11 , ack: 0×00 , move: 0×20)
4. Length: the length of the payload. It makes each message distinguishable from the next one.

Table 1 Overlay message list description

Register	Registers a UNR with a CSOA server. The CSOA server keeps track of the available nodes to serve any incoming query
Query	Used by a UNR and CSOA servers asking by the overlay for available nodes to help an overloaded one
Response	Used by a responder CSOA server when a received query can be satisfied
Move	Used by the requesting UNR to submit the appropriate application and configuration to the discovered node
Ack	Used by both nodes and servers to positively/negatively respond to a request. In case of registration it also transfers the Neighbour List of the registrar server

Fig. 2 CSOA communication sequence diagram**Table 2** General CSOA message header structure

Header				Payload
Message ID: 16 byte	TTL: 1 byte	Type: 1 byte	Length: 4 bytes	...

- Some fields are used by several message types:
- Port: (2 bytes) the listening port of the message originator at which an incoming connection can be accepted
- IP: (4 bytes) the IP address of the message originator
- NT: (2 bytes) the network throughput that a node requests or makes available
- PT: (2 bytes) the processor throughput that a node requests or makes available

The structure of each message's payload is shown in Table 3.

Table 3 CSOA message payload description

Type	Fields	
0 × 01	Port IP NT PT	IP and Port for incoming connections and the maximum shared capabilities of the node/server to be registered with the overlay
0 × 10	Port IP NT PT S:1byte	IP, Port, requested minimum NT and PT of the requestor node's server. The response has to contain nodes each of which has a minimum NT and altogether the PT if S = 0 × 00 otherwise, each one a minimum PT and altogether the NT
0 × 11	Port IP NT PT 1...*	A list of IP, Port, maximum shared NT and PT of the selected nodes
0 × 20	EL:4bytes E FL:4bytes F	The executable E of size EL (in bytes) is necessary for the shared node to participate in the network. Its configuration file F has size FL (in bytes)
0 × 00	A:1byte Port IP 1...*	A = 0 × 01 if the acknowledgement is positive and A = 0 × 00 if negative. If used in reply to a registration request a list of {Port,IP} tuples are followed representing the registrar's server Neighbour List

Server States

Every server is in initialization state when it tries to register to the overlay and in active state when it is in the overlay and reacts to the messages from the overlay and underlying network. The active state can get further analysed to the outgoing and incoming query state.

Initialization State

The initialization of a server when it joins the overlay is equivalent to the initialization of the neighbour list it uses. The new server (unregistered) uses an existing server (registrar) to retrieve the latter's Neighbour List. Both the registrar and unregistered servers have to record each other as one of their neighbours by rejecting randomly another one if their Neighbour List is full. The unregistered one sends an activation message to the registrar. The latter responds with an activation message, which contains its own Neighbour List. The unregistered server uses this list as its own Neighbour List and sends the same activation message to all its new neighbours. It appends to this Neighbour List the neighbours of its primary neighbours until its Neighbour List is full or no more activation messages are received.

The commitment of the registrar and all its neighbours to record the unregistered server as their new neighbour reduces the possibility of fragmentation of the overlay. This registration process ensures the low cost of the overlay initialization and expansion.

Outgoing Query State

The underlying node constructs and sends the appropriate query to the local server. The Node Pool responds to the Query Processor with an empty answer, if no nodes can be found, or, otherwise, a list of nodes that satisfy as much of the requested capacity as possible. If the query is not completely satisfied by the Node Pool the Query Processor forwards a query to its neighbours requesting for the rest capacity that the local pool cannot provide and reducing the query's TTL by one. Otherwise, a proper answer is sent back to the requestor node.

Before the Query Processor forwards the query it caches it and waits for answers. The cache removes the query when its lifetime ends or when the first answer is received. If an answer for the same query is received and the cache has no record for the query, then this answer is rejected and a negative acknowledgement is sent to the originator.

The first answer that is received by the query originator's server is immediately sent to the Node Pool to reserve the retrieved alongside to the already reserved local nodes. Simultaneously, the query processor sends a positive acknowledgement back to the answer's originator server and a complete answer with the nodes reserved in the local Node Pool back to the query's originator node. The pool, then, waits for a positive or negative acknowledgement by the query originator. If the acknowledgement is positive (the answer is accepted) the reserved nodes are removed by the

pool, otherwise (timeout or negative acknowledgement) all the reserved for that query nodes are marked as free and remain in the local pool.

Negative acknowledgement is sent when there is no need for the answer anymore or if the answer is invalid (e.g., one of the retrieved nodes has failed). Then, the node adds the unsatisfied requested capacity to any new amount of required capacity and reinitiates another query to the local server. Figure 3 presents the flow control when the CSOA reacts to an underlying query.

Incoming Query State

Queries from the overlay are treated differently. Initially, the processor checks if the cache contains a query with the same ID. If such an ID exists, the query has already been processed and therefore no action is taken. Otherwise, the Query Processor caches it and transfers the control to the local Node Pool.

When the pool receives a query, originated from the overlay, it tries to match all the requirements over its resource availability. The query is satisfied only if the resources in the node pool are more than the safety capacity and the query is fully satisfied. The pool satisfies a query even if the response may reduce the resources to a lower than the safety level. The matched resources, if any, are reserved and returned to the Query Processor, otherwise the answer is empty.

Then the server caches and sends the answer back to the query's originator server and waits for the negative or positive acknowledgement of the originator. When a negative acknowledgement is received or the lifetime of the answer in the local cache expires, the answer is removed from the cache and the reserved nodes of the pool are freed. If a positive acknowledgement is received the answer is removed from cache and the nodes of the answer are relocated to their new manager.

In the case where there is not enough capacity in the Node Pool the Query Processor reduces the query's TTL by one and if that is bigger than zero it forwards the query to its neighbours selected by the Neighbour List. Using the same notation, Fig. 4 illustrates the process control flow in case a query has been received from the overlay.

Fig. 3 Process flow for the queries originated from the underlying network

Fig. 4 Process flow for the queries originated from other servers of the overlay

Underlying Network Relocator

The UNR is a proxy server residing in the underlying node monitoring the CPU and network usage of an application. The application is manually registered with UNR which measures the CPU cycles spent and the data volume transferred over the network connection by the application's client on a per time-unit basis.

The UNR can register and create a configuration profile for more than one application. Apart from the application's path, the UNR profiles two tuples (min, max) representing the minimum and maximum allowed CPU and network throughput (CPU cycles and kbits per time-unit respectively) per application. These thresholds are used as metrics to trigger an action (register, query) to CSOA. Under the CSOA perspective, the UNR is the system's client which forwards appropriate messages to CSOA servers monitoring an application.

A registration message is sent to the CSOA server if both the network and processor throughput used by the application are below their corresponding minimum thresholds (underloaded state). If any of these metrics is above its maximum threshold then the underlying node is overloaded and a query message for help is sent to the CSOA server. Table 4 presents the actions a UNR can take based on the state of the two metrics.

The application is normally loaded if the CPU and network throughput consumed by the application do not exceed their maximum thresholds and are not both below their minimum ones.

While the max threshold protects the application host from devoting too many resources into the network it participates, the minimum one improves the host's utility. The used processor or network throughputs are determined based on the activity of the application the last T time units it was active. That is, the UNR records an activity history to calculate the average throughputs.

The application's current network throughput is the sum of the data volume that is received and sent within the current time unit. The application's processor usage can be determined monitoring the processor state.

Table 4 Actions of UNR with respect to the state of the monitored network application

CPU	Network		
	Underloaded	Normally loaded	Overloaded
Underloaded	Register	None	Query
Normally loaded	None	None	Query
Overloaded	Query	Query	Query

The requestor node creates a query message specifying as search criteria the two shared resources, the minimum network and processor throughput. Every CSOA server, receiver of a query, tries to select the minimum subset of nodes. The application may specify one criterion (high priority) that needs to be satisfied by every selected node and the other one (low priority) by all of them. That is, the sum of the selected nodes' shared low priority resource values has to be equal or bigger than that criterion.

Server and Node failures

One of the basic features of an unstructured P2P network is the unreliability of its nodes. In the case of CSOA there is no assumption about the stability of the servers and the underlying nodes. It is assumed that both of them reside in equally unreliable machines and have equally unstable connection.

When a server has failed the links from the underlying nodes and its neighbors are broken. Therefore, no queries, answers or activation messages can be sent from/to that server. If no answer can be sent, neither can an acknowledgement since it is only sent after an answer. All the nodes that are reserved become free and the cache empties when the server returns to active state. It is assumed that the acknowledgement is received immediately after a successfully sent answer.

While the local server is failed, the underlying nodes keep generating queries with a lifetime asking each time for the additional capacity they need. When a generated and not sent query expires, its requested capacity is added to the one of a new query. This saves the overlay from queries that ask a huge amount of capacity when the local server comes back. The nodes that are underloaded try to send periodically an activation message until the local server is able to respond positively.

Similarly, when a free node fails, the Node Pool contains a broken link to it. Therefore, an answer may contain both alive and failed nodes. This is detected in the requestor node and the query is dropped. If the requestor node is failed, the Query Processor cannot communicate to deliver the found answer and thus, the local server frees any reserved node for that query.

Simulations and Evaluation

A simulator is based on the Object-Oriented Programming principles and implemented in C++ to help the evaluation of the proposed architecture. The

simulator is time-based and can execute multiple concurrent queries sent or forwarded by a random number of nodes. It also sets server and node failure rates to simulate the dynamic behaviour of P2P networks. The simulator executes the following three phases in every iteration (time slot):

- *Set global workload:* new workload is set over a random number of underlying nodes on any network. This additional workload is not evenly distributed over the selected nodes but a random amount of it, even negative, is applied on each one. The sum of these partial workloads must be equal to the total additional system-wide workload.
- *Fail servers and nodes:* a preset percentage of servers and nodes fail in every iteration. The simulator selects these servers and nodes randomly between all the servers, including those that failed in any previous iteration. Any server or node that has already failed may be selected to keep failing and if not then this server or node is activated again.
- *Send produced messages:* Both of the phases above produce queries since adding workload on an underlying network in conjunction with failing a number of active nodes may cause network overloading. In this phase all the new queries are sent from their originators and all the past queries are forwarded from the intermediate servers to their neighbours.

Performance Metrics

There are numerous metrics we could use to evaluate the performance of our capacity sharing system. We have selected the following ones:

- *Success Rate:* This is the ratio of the number of successfully answered capacity searching queries over the total number of queries generated in the system. This metric serves as an indication as to how good CSOA is in terms of finding the capacity required by the over-loaded underlying networks. We expect that the value of this metric will initially be very close to 1 as there is plenty of spare capacity in the system and then it will decrease significantly as the whole system will be reaching saturation.
- *Satisfied User Queries:* This is the number of additional user queries that overloaded networks have managed to satisfy because they acquired extra capacity through the CSOA. For example if a network asked for 10 capacity units and the CSOA successfully provided free nodes whose cumulative capacity is 10 or more units then we can say that the CSOA has made it possible for 10 more user queries to be processed successfully. This metric shows directly the main benefit of our proposed system. We expect the value of this metric to depend heavily on the success rate.
- *Hops per Answered Query:* This is the number of hops a query performed until the first valid answer is returned. It indicates how quickly the system finds answers to capacity queries and we expect it to increase, as the whole system will be reaching saturation.

- *Messages per Query*: This is the number of messages a capacity query generated while it was being forwarded within the CSOA. This metric, in conjunction with the previous ones, will help us evaluate the performance and economy of different blind search strategies supported in CSOA.

Environment Characteristics

To use the above metrics for the CSOA evaluation we had to build a model of a typical representative environment within which CSOA could be deployed. The modelled environment has the following characteristics:

- There are 1,000 independent networks each of which is connected to the CSOA overlay through a single representative server. Therefore, there are 1,000 CSOA enabled servers.
- Each network has 50 nodes of which only one is initially active within the network with the remaining being free. Therefore, the system has 50,000 nodes; a small number of which are initially active.
- Each node has a random network capacity up to 20 units. Because we used a uniform random number distribution the system has in total a capacity of $\sim 500,000$ units.
- Each node has a random list of 4 neighbours. The Time-To-Live (TTL) of every capacity query is set to 3.
- The simulator executes 1,000 iterations. In every iteration, the workload in the whole system is incremented linearly from 0 to the theoretical maximum of 500,000 units. The workload is distributed to a random, in every iteration, sequence of nodes following a normal (mean = 4, variance = 1) distribution until it is consumed.

Each node that produces a query sends the query to the next time slot of the simulation. In the meantime, the node may fail and its workload is randomly fragmented and distributed to the rest nodes of the underlying network. This phenomenon may be negligible in certain underlying network topologies if there is no node that can take over the workload of the failed node. In this case, the query does not leave the node and the node returns to the server's Node Pool when it is active again. If the node manages to send the query to the local server and then fails before an answer is received the server detects that the node has failed since it cannot return the answer and therefore, places any retrieved from the overlay node into its Node Pool. This may work as a proactive mechanism for the next queries from the same underlying network.

Simulation Results

The two searching techniques that CSOA supports (flooding and k-walkers) were simulated under various failure rates (0, 2, 6, 10 and 14%). The failure rate is applied to the whole system; both servers and nodes have the same probability to fail in every iteration. The results shown below are grouped into four categories of

experiments, one for each evaluated performance metric. Each category has four experiments for the evaluation of the metric, first, in case of no failures on all the two simulated searching techniques and then with failure rates of 2, 6, 10 and 14% on the flooding, 2-walkers and 4-walkers techniques separately.

Success Rate

From Fig. 5, we observe that the flooding technique has 100% success for longer time compared to the k -walkers ($k = 2, 4$). This is because it is more aggressive in finding resources. However, it consumes resources faster; therefore, when capacity becomes scarce, it drops more abruptly than k -walkers. From Figs. 6, 7 and 8 (failure rates 2, 6, 10 and 14%) we observe that all these three techniques experience a drop in their success rate sooner in simulation time compared to the experiments with no failures. We also observe that as the failure rate increases, all three techniques appear to have better success compared to lower failures. In fact, the graphs tend to become more horizontal and significantly scattered. There are mainly two reasons for this phenomenon:

1. As the failure rate increases, it means more queries timeout or are dropped as an increasing number of CSOA servers fail regularly. In this way, in the first part of the experiments when there is plenty of free capacity, fewer queries are satisfied; thus more capacity is to be satisfied in the latest stages of the simulation. Although, initially we have significant reduction of success rate, especially for k -walkers, compared to the no-failures experiments, towards the end of the simulations we have a significant increase as we also have more available capacity than in the no-failures experiment.
2. The scattering phenomenon of the graphs is also due to the fact that the more failures in the system the more random and chaotic it becomes to find an answer

Fig. 5 Success rate of flooding and 2/4-walkers without node failures

Fig. 6 Success rate of flooding with node failures

Fig. 7 Success rate of 4-walkers with node failures

and therefore there is significant fluctuation in the success rate, even between consecutive timeslots.

Satisfied User Queries

The experiments in the Fig. 9 repeat the same phenomenon as in the Fig. 5. Flooding consumes faster the resources finding for longer time more capacity than the k -walkers but drops more abruptly when these resources become scarce. From the experiments in this section, we observe that the satisfied user queries drop at the

Fig. 8 Success rate of 2-walkers with node failures

Fig. 9 Satisfied user queries using flooding and 2/4-walkers without failures¹⁸

same point as the success rate of the figures in Sect. "Success Rate". From Figs. 10, 11 and 12 (with failures), we observe that the satisfied user queries are many more than in the experiments with no failures. This is the effect of the servers' failure. As a server fails, all the nodes of its underlying network cumulate the capacity of the queries that cannot be sent because of its failure. This accumulated capacity is queried when the server is again up and running. Therefore, more capacity is requested and the capacity of the overlay is consumed faster. As the failure rate increases the servers that have failed in one timeslot have increased probability to be selected in the next timeslot to remain failed, thus, deteriorating the phenomenon.

Finally, the higher failure rates cause more nodes to fail, too. Since the failure rate and network size remain fixed throughout an experiment, the ratio of the nodes

Fig. 10 Satisfied user queries using flooding with node failures

Fig. 11 Satisfied user queries using 4-walkers with node failures

that become active in each iteration is also fixed and increases with the failure rate. That way, more available resources are produced with the increase of the failure rate. The inability of flooding to serve capacity in the later stages of the simulations gets also worse due to the large number of reservations it does.

Hops per Answered Query

The flooding effect that causes the fast consumption of the resources of the overlay also forces the queries to travel deeper in the overlay compared to the k -walkers (Fig. 13). The low success rate, after that point, causes few queries to be answered

Fig. 12 Satisfied user queries using 2-walkers with node failures

Fig. 13 Average hops per answered query of flooding and 2/4-walkers without node failures

and therefore, the number of those that got answered within less than TTL steps is not negligible causing the scattering effect in the end of the Fig. 13 experiments. From Figs. 14, 15 and 16 (with failures), we observe that the flooding needs more hops to find the required resources than the k -walkers. This is another side-effect of flooding, since for every answer found, the nodes are reserved until an acknowledgement is returned to the answer originator. The answer is again reserved in the query's originator server until it reaches the query's originator node. Flooding causes more reservations than k -walkers reducing the capacity availability and thus increasing the hops of the queries.

Fig. 14 Average hops per answered query of flooding with node failures

Fig. 15 Average hops per answered query of 4-walkers with node failures

An interesting phenomenon is that the bigger the failure rate the smaller number of steps required for the queries. This happens because in high failure rates more servers fail; thus more queries are dropped before they expire and less capacity is consumed. Moreover, the higher the failure rate the more nodes fail and the more nodes return to their local Node Pool when they become active again. When a node fails, its workload is distributed to the other peers of its underlying network reducing the probability of overloaded situations in the network; thus, fewer queries are generated. Finally, if the node has already sent a query and then fails, the nodes that are retrieved are placed in the local Node Pool since the node does not need them anymore. This proactively fetches capacity from the overlay for the next

Fig. 16 Average hops per answered query of 2-walkers with node failures

queries. Therefore, not only new free nodes appear in the Node Pools but also fewer queries are forwarded to the overlay and workload of the failed ones is better redistributed over the members of the failed node's underlying network.

Messages per Query

In the case of no failures, Fig. 17 clearly shows the high cost of flooding compared to the 2/4-walkers. The number of messages generated by flooding stops increasing when the success rate of the technique drops abruptly. This can be explained by Fig. 13, which shows the flooding to reach the maximum query TTL after that point.

Fig. 17 Message cost of flooding and 2/4-walkers without node failures

Fig. 18 Message cost of flooding with node failures

No more messages can be generated since the queries cannot travel deeper in the overlay than TTL. The phenomenon is repeated to Figs. 18, 19 and 20 (with failures): the good success rate of flooding costs many messages to the overlay. Finally, the higher the failure rate the lower the cost in messages of the three techniques. This happens because the number of required hops per answered query is smaller, as we explained in Fig. 13.

From the experiments, we understand that through the CSOA a lot of requested capacity from the underlying networks is served even in high failure rates. The absence of the CSOA would cost the networks a lot of unsatisfied user queries and frequent overloaded situations. Moreover, the experiments show that though the

Fig. 19 Message cost of 4-walkers with node failures

Fig. 20 Message cost of 2-walkers with node failures

flooding has a better success rate than k -walkers ($k = 2, 4$) even in high failure rates, it has a much bigger cost for the overlay in messages.

Conclusions

Network applications are dynamic distributed computing environments that frequently exhibit phenomena of under-loading and over-loading due to the massive fluctuation in the number of user queries they produce and handle. For this reason it is beneficial to provide mechanisms that allow over-loaded networks to utilise the free capacity of under-loaded ones. In this way we can achieve significantly enhanced performance in terms of the number of the user queries that can be processed.

The main innovation of CSOA is the concept of logically moving free nodes on demand from one network to another in order to serve extra workload in terms of user queries. These “mobile” nodes also have the autonomy of removing themselves from the network they serve if they are under-utilised and make themselves available for other over-loaded networks. The resulting system is simple in design, deadlock-free (as the system does not reserve partial answers as each query propagates through the overlay) and fully supports heterogeneity as it makes no assumption regarding the topology and type of the underlying networks. Therefore, the system has good applicability in scenarios where instead of network capacity the underlying networks share processing power or even storage capacity.

Through extensive simulations, we have shown that CSOA manages to find the required spare capacity with high probability and as a result, there is a significant increase in the number of additional successfully processed user queries that would otherwise have been dropped. CSOA supports standard blind P2P search techniques such as flooding and k -walkers and it can easily be extended to use others.

The CSOA performance gains remain significant even in the presence of non-negligible failure rates of the CSOA servers and nodes.

An extensive survey from Streamcheck Research [21] revealed that an increasing number of video streaming providers suffer from flash crowds especially in peak hours. For instance, in top news providers like CNN, MSNBC and ABC, because of traffic bursts, the “breaking stories are often unreachable during peak viewing hours” and the users experience content outages and longer startup time. BBC announced that on 7th July 2002 the download time of webpages was eight times longer, a quarter of requests for webpages were not served and “email traffic reportedly doubled in response to congestion on mobile networks” [22].

The high cost of the streaming servers and video repositories as well as the high bandwidth requirements of live or on-demand streaming services may become a substantial burden for several content providers. CSOA can dynamically allocate and use the existing infrastructure of companies that build different applications on different networks. Using a CSOA server per application, capacity can be shared from one network to another based on each application’s requirements.

We have already started to explore a number of interesting avenues for further work. Firstly, we are researching the possibility of designing a capacity search algorithm that dynamically adapts between flooding and k-walkers in order to reduce the number of messages needed until an answer is found. Secondly, we will be looking into ways of incorporating a basic access control layer into our system that will enable the owners of the nodes to specify conditions requestors need to satisfy before they could be given access to them. This access could also be constrained in terms of time or other configurable parameters. Overall though, CSOA in its current stage effectively provides a novel and simple method for sharing processing resources among a high number of networks and therefore it constitutes one more step towards the evolution of P2P networks from file/content sharing systems to high performance, general-purpose parallel processing environments.

References

1. Bedrax-Weiss, T., Macgann, C., Ramakrishnan, S.: Formalizing Resources for Planning, PDDL03: Proceedings of the Workshop on Planning Domain Description Language, Trento, Italy, pp. 7–14, June 2003
2. Cohen, E., Shenker, S.: Replication strategies in unstructured peer-to-peer networks. *ACM SIGCOMM Comput. Commun. Rev.* **32**(4), 177–190 (2002)
3. Tsoumakos, D., Roussopoulos, N.: Analysis and comparison of P2P search methods. *ACM 1st International Conference on Scalable Information Systems*, Hong Kong, China, vol. 152, No. 25, May 2006
4. Yang, X., de Veciana, G.: Performance of peer-to-peer networks: Service capacity and role of resource sharing policies. *Perform. Eval.* **63**(3), 175–194 (2006)
5. Zhou, D., Lo, V.: Cluster Computing on the Fly: Resource Discovery in a Cycle Sharing Peer-to-Peer System, pp. 66–73. *CCGrid: IEEE International Symposium on Cluster Computing and the Grid*, Chicago, Illinois USA (2004)
6. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content Addressable Network, pp. 161–172. *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, California, United States (2001)

7. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for Internet applications. *Trans. Network. IEEE/ACM* **11**(1), 17–32 (2003)
8. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lect. Notes Comput. Sci.* **2218** (2001)
9. Zhao, B.Y., Kubiawicz, J.D., Joseph, A.D.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. University of California at Berkeley, Berkeley, CA (2001)
10. Lua, K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials. IEEE* **7**(2), 72–93 (2005)
11. Litzkow, M.J., Livny, M., Mutka, M.W.: Condor—a hunter of idle workstations, 8th International Conference on Distributed Computing Systems, pp. 104–111 (1988)
12. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. *Concurr. Comp.: Pract. Exp.* **17**(2–4), 323–356 (2005)
13. 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 DUT-TWI-93-104. Delft University of Technology, The Netherlands (1993)
14. Epema, D.H.J., Livny, M., van Dantzig, R., Evers, X., Pruyne, J.: A worldwide flock of Condors: load sharing among workstation clusters. *J. Future Gen. Comp. Syst.* **12**(1), 53–65 (1996)
15. Butt, A., Zhang, R., Hu, C.: A self-organizing flock of Condors. *J. Parallel Distr. Comp.* **66**(1), 145–161 (2006)
16. Androutsellis-Theotokis, S., Spinellis D.: A survey of peer-to-peer content distribution technologies. *ACM Comp. Surv. (CSUR)* **36**(4), 335–371 (2004)
17. Philippe, K.: P-Grid: a self-organizing structured P2P system, sixth international conference on cooperative information systems (CoopIS 2001). *Lect. Notes Comp. Sci.* **2172**, 179–194 (2001)
18. Caviglione, L., Davoli, F.: Peer-to-peer middleware for bandwidth allocation in sensor networks. *Commun. Lett. IEEE* **9**(3), 285–287 (2005)
19. Exarchakos, G., Salter, J., Antonopoulos, N.: Semantic Cooperation and Node Sharing Among P2P Networks. *Proceedings of the Sixth International Network Conference (INC 2006)*, Plymouth, UK (2006)
20. Salter, J., Antonopoulos, N.: ROME: Optimising DHT-Based Peer-to-Peer Networks, pp. 699–702. *Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'05)*, Monte Carlo Resort, Las Vegas, Nevada, USA (2005)
21. Streamcheck Research: News Video Quality Survey, Accessed: 20-02-2007, RefType: Electronic Source <<http://www.streamcheck.com/sc-newsvideostudy.pdf> >
22. News sites toil as visits rocket, Accessed: 20-02-2007, RefType: Electronic Source < <http://news.bbc.co.uk/1/hi/technology/4663423.stm>>

Author Biographies

Georgios Exarchakos is a Ph.D. student in the Department of Computing at the University of Surrey. His research interests include network management and resource discovery in peer-to-peer networks.

Nick Antonopoulos is currently Lecturer in the Department of Computing at the University of Surrey. His research interests include emerging technologies such as web services and peer-to-peer networks, and software agent architectures and security.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.