WOBCompute: Architecture and Design Considerations of a P2P Computing System

Levente Filep[a]
Faculty of Mathematics and Computer Science, Babeș-Bolyai University, Mihail Kogălniceanu, Cluj-Napoca, Romania

Keywords: Peer-to-Peer Networks, Super-Peer Topology, Distributed Computing, Middleware.

Abstract: Regarding large-scale scientific computing, many alternative solutions to Cloud Computing Services exits, which combine existing, cheap, commodity hardware into computational clusters. The majority of these, due to their ease of deployment, are based on Client-Server architecture. Decentralized approaches employ some form of Peer-to-Peer (P2P) design, however, due to their increased design complexity, and without major benefits over the Client-Server ones, none of these systems gained wide popularity. The P2P system presented in this paper features decentralized task coordination, the possibility of suspending, migrating and resuming workload on different nodes, employs remote checkpoints to allow partial result recovery, and workload tracking, which offers the possibility to initiate communication between them. Design considerations and choices for this system are presented and discussed. The chosen topology is super-peer managed clusters arranged in an extended start topology and evaluated by simulation. Such a system comes with enormous design complexity; however, a middleware can hide these complexities, while providing the applications a simple interface to access network resources. Harnessing idle computing resources, the system can be deployed on a combination of in-house computer networks, personal and volunteer devices, as well as Cloud-based VMs.

1 INTRODUCTION

In recent years Cloud Computing, in the form of Cloud Services, has become the defacto choice for large-scale scientific computing due to its availability and low cost. Despite this, there are many instances where this is out of reach for individuals or small research groups.

Many alternative solutions exist that combine existing, cheap, commodity hardware into computational clusters, such as grid computing or volunteer computing (Lavoie and Hendren, 2019). The latter is achieved by utilizing a specialized middleware to harness the idle computational resources of volunteers. Such a middleware can also be deployed in a combination of personal devices, in-house computer networks, and even Cloud VMs. Existing solutions are mostly centralized in nature, based on Client-Server architecture, where the centralized server(s) are responsible for task coordination (creation, deployment and result collection), while volunteer resources are utilized to run these tasks. Due to their ease of deployment, this kind of system, such as BOINC (Anderson, 2004), has become the most widespread. Decentralized approaches employ some form of Peer-to-Peer (P2P) architecture, where each node takes part in both the execution and the coordination of tasks. However, P2P solutions have an increased design complexity compared to the simplicity of Client-Server based systems, and without providing additional benefits (Lavoie and Hendren, 2019), these never gained popularity, despite having the advantage of decentralized task coordination, which, in case of large projects, eliminates the need for costly, high-performance centralized servers.

In this paper, a new P2P system is presented; based on a previously presented model (Filep, 2019), which is characterized by decentralized task coordination (creation, deployment, and result dissemination) as well as task tracking and remote backups, which allows recovering partially completed computation. Task tracking, in other words, the ability to query task location, brings the

[a] https://orcid.org/0000-0003-2095-0161

Filep, L.
WOBCompute: Architecture and Design Considerations of a P2P Computing System.
DOI: 10.5220/0009343100390049
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 39-49
ISBN: 978-989-758-421-3
Copyright © 2020 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

possibility of parallel-branch communication. Such a system comes with enormous complexity, however, employing a specialized middleware, these complexities can be hidden from the application. As the middleware harness idle resources, the system can also be deployed on a combination of in-house computer networks, personal and volunteer devices, as well as Cloud-based VMs. Design considerations and choices regarding the system's topology and middleware are also present and discussed. Application design for such a computational system is discussed in a different paper.

2 RELATED WORK

P2P systems are commonly used nowadays in data networks, such as data sharing or streaming. As the basis of computational networks, the P2P architecture has not gained a wide interest in the literature, arguably due to their design complexity. In Grids and Volunteer Computing (VC), where the parallel tasks can be computed independent of each other, the client-server architecture is the popular choice for such systems, as in the before-mentioned BOINC (Anderson, 2004).

2.1 P2P Computing Systems

P2P architecture also influenced several computing system designs. CompuP2P by Gupta et al. (2006) is a lightweight architecture for internet computing, using node clustering, called markets of computing resources. Tiburcio and Spohn (2010) presented a P2P based open-source computing grid. Gomathi and Manimegalai (2013) presented the Hierarchically distributed Peer-to-Peer (HP2PC) as a solution to heterogeneity problems. Pérez-Miguel et. al. (2013) presented a prototype P2P-HTC (P2P High Throughput Computing) system based on Cassandra (a distributed DHT based database) for distributed queue based scheduling utilizing an FCFS (First Come First Served) scheduling policy. DisCoP (Castella et. al., 2015) is also a P2P system that harnesses idle CPU cycles and a clustered topology for resource location. These implementations, however, didn't gain huge popularity when compared to BOINC.

2.2 Virtual Topology

P2P networks create a virtual topology over the physical one, which directly results from the set of rules the nodes use to connect between themselves. Such an overlay can be structured or unstructured. Structured P2P networks maintain a virtual topology on top of the physical network layout (Ratnasamy et al., 2001) aimed at increasing reliability, availability, and search speed. Chord (Stoica et. al., 2001) employs a ring topology and uses finger tables to improve search efficiency. Tapestry (Zhao et al., 2004) introduced the concept of backup- neighbor to maintain the virtual overlay if a node becomes unavailable.

The concept of super-peer with redundancy to maintain cluster stability was proposed and evaluated by Yang and Garcia-Molina (2003). However, the super-peers concept by itself introduces a single point of failure in the network. In the before-mentioned paper, the authors also used redundancy and concluded that this does not significantly affect the overall bandwidth usage. They also proposed cluster splitting for the network topology to adapt to an increasing number of connecting nodes and cluster merging in case of decreasing node number when some clusters become too small; however, no exact solution was proposed on how to perform these two operations.

We often think of P2P systems as where all nodes are equal; however, each node differs regarding their computational performance, storage capabilities, and available bandwidth (Jesi et. al., 2006). In the SG-2 protocol (Jesi et. al., 2006), a proximity-based super-peer election is proposed, where each node has an associated latency distance. The authors also state that due to the constant change in the peers, the maintained overlay is highly dynamic.

Clustering of peers as a search improvement was proposed by Ye Feng et al. (2009) for the Gnutella protocol, but also as recent as by Vimal and Srivatsa (2019) for file sharing system search.

Several studies have analyzed the reliability of super-peer based P2P networks. A study by Mitra et al. (2008) finds that a super-peer ratio of less than 5% sharply decreases reliability. Super-peer based networks are also vulnerable to churn (De et. al., 2016), where a large number of simultaneously connecting or disconnecting nodes can divide the network into isolated parts. In such cases, data replication was proposed by Qi et al. (2019) as a solution to data survivability.

3 SYSTEM ARCHITECTURE

WOBCompute is a P2P based computing system based on a previously published model (Filep, 2019).

We can briefly summarize this model by its characteristics:

- Workload (or task) creation is offloaded from centralized servers to the peers; each workload starts as a whole, and it is split based on other peer's requests for computation while also accounting for their performance. In this case, the application itself is responsible for splitting the workload, which allows for multiple types of workloads, offering better load balancing.
- The completed child workload result is transferred to the parent task. The dissemination process is also controlled by the application itself.
- Gateway(s) are used for the overall workload injection and result collection.
- Computation of a workload can be suspended, transferred to a different node and resumed.
- Workload locations can be queried by the application for initiating communication between them. The connection is opened directly to the target node or via a super-peer if the target is behind NAT.
- The use of periodic remote checkpoints allows the recovery of a partially computed workload, which then can be resumed and even further split. The application can choose whether to wait or recover a workload.

To achieve the above objectives, the previously presented model extends the notion of workload unit with additional data fields: unique identifier (ID), application identifier, parent and children identifier (for result merging), checkpoint data (allows the transfer and continuation of computation), boundary information (identify data-set boundaries; if required), estimated total and remaining computational effort, state of computation, result data (contains the partial result of the workload) and metadata (for any application-specific use).

For ease of processing, transfer with minimal bandwidth usage, and storage, the above fields are incorporated into a single data-object represented as a JSON structure, named Workload Object or WOB in short. Furthermore, WOB size is to be kept at minimal, therefore large analyzable data-sets are to be acquired separately.

The concept of WOB based computing allows the system to incorporate in-house computer networks with volunteer provided resourced and Cloud-based VMs. Furthermore, the possibility to track WOBs opens possibility to initiate communication between the parallel branches of an application. Due to arbitrary latency between nodes, such application design must be latency tolerant.

3.1 Network Topology Design

As mentioned before, P2P networks create a virtual topology over the physical one. Starting from the base model of the system, we need a search protocol to query each WOB and a distributed storage for the WOB backups. There are a variety of implementations for both distributed search and storage, however, in the author's opinion, having two more virtual overlays on top of the one created by the middleware, would significantly increase the system's complexity.

DHT based systems have proven to be the fastest when it comes to search, especially when dealing with rare resources. In the presented system, each WOB can be considered rare; however, due to constant migration, creation, and dissemination of them, especially with a large number of objects, the number of update messages may significantly surpass those of query messages. This can lead to significant bandwidth and resource consumption to keep the DHT up-to-date. It must be noted, that the above statement is an assumption and was not tested experimentally, however, to the author's best knowledge, how DHTs behave with extremely frequent keys and value changes was not examined in the literature.

WOB creation is driven by workload request messages and therefore must reach all nodes. The number of these messages can be problematic as a WOB nears completion, in which case as more and more nodes become starved, a huge number of messages can overload and cripple the network. Clustering of the nodes offers a pretty straightforward solution: a cluster that contains starved nodes has no reason to accept any workload request messages. Furthermore, if a super-peer is aware that no more workload is available, they can stop all outbound request messages to other clusters. However, if more workload becomes available (e.g. a new WOB is injected via the gateway), workload notification messages can unlock clusters and trigger the idle nodes to request workload.

Several topologies, other than the previously mentioned ones, have been proposed and demonstrated to be resilient and efficient in resource location, such as the AFT (Poenaru, 2016). To best of the author's knowledge, besides clustering with super-peers, none can offer a cutoff solution to the above-mentioned message flooding problem. Furthermore, super-peers can act as a tracker of the WOBs located within their cluster, thus search queries can be limited only to them. A similar proposal was made by Chmaj and Walkowiak (2013),

where, as opposed to the current design, the tracker is also responsible for task scheduling.

For the above-discussed reasons, the chosen topology is the super-peer driven clusters of peers, arranged into an extended start superstructure, where each cluster is connected to their parent.

Super-peers, given their role, will be referred to as supervisors. The cluster supervisors keep a list of all WOBs located within their cluster and their latest backup. This way, the search only involves the supervisors. Backup supervisors also keep a copy of the WOB list and backups. Furthermore, they also keep track of the available number of workload present and idle nodes required for limiting workload request messages.

The chosen topology does not exclude the possibility of a later DHT implementation on top of the current topology, where only the super-peers would participate. Their redundancy would reduce the number of joins into and disconnects from the DHT topology, as one would leave, a backup super-peer with already up-to-date indexes would take its place.

As a project can be run on a combination of in-house computer network, outside volunteers and cloud resources, each project has its gateway, thus each project has a separate logical topology. This is required to separate the workload to maintain the usage of allocated resources within the target project. However, this does not exclude the possibility of one peer participating in multiple projects.

Due to network latency between the nodes, we can notice that while a task can initiate communication between parallel branches of an application, there are limitations to the types of applications that can benefit from such a design. In other words, the efficiency of an application decreases as the number of messages in given time-interval increases; with an increased number of messages, an application will spend an ever increased time on communication instead of computation.

3.1.1 Cluster Topology Considerations

The search efficiency is considerably impacted by the number of clusters, meaning, the greater their number, the more hops a query message has to take to reach all of them. Reducing the cluster count and still connect the same number of nodes can be accomplished by increasing the number of nodes a cluster accept. Since each node has a limited connection capacity, the cluster members are distributed among the supervisors. We can define a cluster as balanced when the load ratio on all supervisors is about equal. Balancing operations can be triggered if a supervisor is overloaded.

In the current topology, each regular node is connected to only one supervisor but is aware of all other supervisors, as illustrated in Figure 1. In contract, all supervisors are connected between them. As a synchronized list of WOBs and their backups are maintained, the persistent connection makes messaging easier.

Node failure detection is accomplished through regular heartbeat messages. If the connected supervisor fails or leaves the network, the affected nodes will connect to another supervisor from the cluster. If this also fails, we consider the node isolated, which then re-joins the network through the gateway.

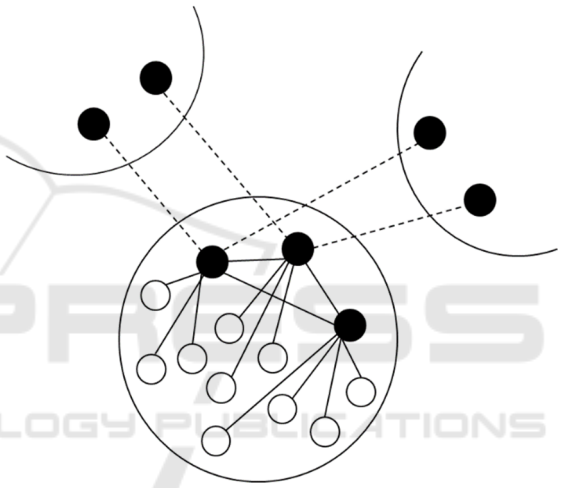

Figure 1: Cluster topology.

Changes in the available supervisors, such as disconnects or elections, are advertised throughout the cluster, so each node has an updated list of these.

3.1.2 Topology Stability

In the present system, WOB query misses are not allowed as they can lead to computation losses. For instance, if a WOB is partially computed, but the query misses it due to isolated clusters, the querying node may assume it to be lost and choose to re-create it from a backup. Therefore, topology stability is paramount.

Clusters are interconnected with the help of two supervisors, one from each cluster. For additional reliability, a backup connection is also maintained; however, messages are not balanced between the two connections. As a backup scenario, if a cluster gets isolated from its parent, it will query the gateway for

a list of current clusters and attempt to connect to one. This offers some stability in case of churn.

As the system harnesses idle cycles, when a node is not participating in any computation, meaning its compute resources are not free, they can still participate, by user choice, in basic interconnection functions. This in term further increases topology stability.

3.1.3 Cluster Stability

Cluster internal stability is also an important factor. If a supervisor fails, the connected nodes try to connect to one of the known supervisors within the same cluster. If those get overloaded and refuse the connection, then we get isolated nodes; as the election process of a new backup supervisor, followed by balancing the existing supervisors can be more time-consuming than the re-joining process of the node.

If a cluster becomes unviable (shrinks to very low node number) we can consider it defunct. In this case, the stored WOBs and WOB backups must be transferred to either a better-suited neighbor cluster or the gateway. We can notice that the system requires a minimal constant number of participating nodes to remain stable. Having a large number of nodes that all disconnect a certain time of day, for example at night, the gateway must act as storage for all WOBs, otherwise, if nodes holding WOBs will not join the network again, then significant parts of the overall computation can be compromised, as child WOB results cannot be disseminated without their parents.

Keeping a reserved connection capacity for each supervisor reduces the chances of isolated nodes occurring if a supervisor leaves the network. Furthermore, supervisors can still accept connecting nodes into the cluster by using their reserved capacity. We can define the total connection capacity allocation of a supervisor as:

$$C_T = C_S + C_R + C_N \quad (1)$$

where C_T denotes the total capacity (minus the inter-cluster connection if present), C_S the number of connections to all other supervisors within the cluster, C_R is the reserved capacity, and C_N the capacity available for cluster member connections. Since we balance the cluster members among the supervisors (primary and backup), we can observe that the cluster capacity can dramatically increase using this technique, but we also notice that there is an upper limit on the cluster size.

With a uniform capacity of 100 (fairly regular setting on BitTorrent clients) on all nodes and a C_R value of 0.1 of C_T, as illustrated in Figure 2, 45% of a

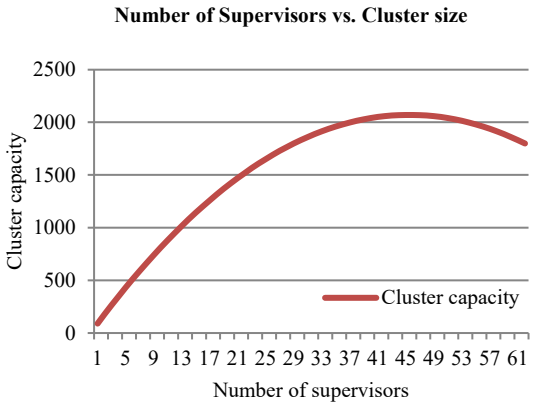

Figure 2: Number of supervisors influence on cluster capacity.

supervisor capacity allocated to super-peer connections produces the optimal cluster size, namely, 2070 of possible connected nodes.

Simplified, we can state that an optimal number of supervisors per cluster are 45% of the lowest supervisor capacity (C_T). A newly elected supervisor connects to all other supervisors, and by doing so, in the handshake process it advertises its capacity, so the maximum number of supervisors can be adjusted accordingly after each election. However, electing a supervisor with low capacity should be avoided as it triggers the demotion process, which removes the lowest capacity supervisor first.

The connections in Equation 1 represent the persistent ones. Temporary connections, such as workload related, are not accounted for as they don't have a significant impact here.

Supervisor Promotion. Supervisor election is triggered when all supervisors exhaust their capacity. The process is trivial: we select the best node based on its capacity and bandwidth. Having a reserved capacity on each supervisor allows the cluster to still accept connections while the election process is running, despite the cluster being temporary overloaded.

Supervisor Demotion. Having too many supervisors is not necessarily a problem; however, maintaining the WOB indexes and backups utilizes an update message for each occurring change, which must reach each supervisor. To minimize the number of these messages in the process, if the average used supervisor capacity drops below a threshold value, the lowest capacity supervisor is demoted. The threshold value is a subjective choice with little impact other than the number of update messages.

Cluster Splitting. When a cluster gets overloaded, and the maximum number of supervisors has been reached, the splitting process is triggered, where randomly selected supervisors along with the connected nodes are split into a new cluster, while the current cluster becomes its parent. The WOB indexes and backups are also adjusted accordingly. As the new cluster becomes functional, the gateway is notified of this.

WOB List Consistency and Backups. A node getting temporary isolated isn't a critical issue as the WOBs under processing still resides on it, and computation can continue even if the node is temporarily disconnected from the network. Furthermore, the node address doesn't change, so workload related operations can still be carried out. For this reason, once the node is detected missing, the WOB entry at the former cluster isn't immediately discarded, but marked as "outdated" and only later removed. This marking prevents conflicting results of the location query; for instance when the node joins another cluster where it advertises it's WOB. In such a case the query of the WOB will return one "outdated" result and one "fresh" which overrules the first one. In the worst-case scenario, if the node didn't have time to join another cluster, then the "outdated" result will still correctly indicate the WOB location.

As network time may be unreliable, WOB backups are also stamped with an incremental number.

When a WOB is completed, and its result merged with the parent, the list entry and backup of the WOB is removed. This minimizes the supervisor index list and the amount of storage required to track the WOBs. The removal process is triggered by the parent node's ACK message of the child WOB.

3.2 Communication and Messages

Node interconnection utilizes TCP protocol and messages are represented in JSON structure. To reduce bandwidth utilization, data communication is compressed using ZLIB (Gailly and Adler, 2002) streams. The use of compression is determined in the handshake process between any two nodes.

A generic message contains the fields presented in Table 1 but, depending on their type, some can be empty or omitted.

Table 1: Message wrapper structure.

ApplicationID	Distributed app unique ID
Message UUID	Unique message identifier
MessageType	Type of message
SenderID	Sender node unique identifier
SenderAddress	Sender node IP:Port
DestID	Destination node unique identifier
DestAddress	Destination node IP:Port
Relayed	Set to 1 if message was relayed by a super-peer
Payload	Contents of message

Separate node ID and address is used for relaying messages through supervisors, but to also identify nodes between IP address changes.

For fast message routing within the middleware, the design-choice was to prefix the message types according to their types. These are:

- WOB messages, prefixed "WOB_". These include the request, response offering, query, backup query, transfer, location update, etc. and their appropriate acknowledgment (ACK) messages.
- Peer messages, prefixed "NODE_", which include interconnection, handshake, and capacity advertising messages.
- Cluster specific messages, prefixed "CL_", include supervisor election, cluster balancing and splitting, but also WOB list and backup update, and cluster interconnection messages.
- Application-specific types, are prefixed "APP_", and are passed to the application itself. These can be used for messaging between different tasks.

3.3 Gateways and Bootstrapping

As dynamic task coordination eliminates the need for centralized servers, a dedicated server is still required to ask a gateway for bootstrapping the network, the injection of a primary task, and computation result extraction. Another role of the gateway is that of failsafe storage for WOBs in case of cluster dismemberment. Since the gateway has a simplified function and no persistent connection is required to be maintained to it by any supervisor, this can easily be implemented as a web application.

Bootstrapping is the process of creating the initial network overlay to which the rest of the nodes join. The first joining node creates the first cluster to which the rest of the nodes join. The join process consists of the following steps:

```
Node contacts the Gateway to obtain the
   list of optimal clusters (cluster id
   with multiple supervisors)
```

```
If the Gateway is unreachable, attempt
   to reconnect to the previous cluster
   (if any)
From the obtained list, chose the best
   one (based on latency)
Attempt to connect to cluster (try each
   supervisor until one accepts the node)
If connection to all supervisors fail,
   try next cluster; if all clusters
   fail, repeat the process
```

As query time depends on the number of hoops the message takes, the choice is to grow the existing clusters to their maximum size before creating new ones. This strategy can also be applied to maintain the central cluster in the topology: if the member count drops, the gateway will prioritize connecting nodes to this. To perform such strategies, clusters supervisors regularly update the gateway about their status.

While messages can be delivered to nodes behind NAT, they are not allowed to become super-peers. Theoretically, they could form a “local” cluster, but no external connection could be initiated to them for maintaining the overall network topology. On the other hand, this imposes a limitation on network bootstrapping, namely, it only can be started with nodes that have an external IP address.

3.4 Middleware

The middleware plays an essential role in maintaining the network structure, providing the required functionalities (such as WOB indexing and backup storage), but also by hiding the complexity of the system, and provides a simplified way for application developers to harness resources from across multiple network nodes. Nonetheless, the utilization of middleware results in additional communication and computing overhead.

The communication overhead is the result of all in- and outbound messages being routed through the middleware. Given the JSON representation of the WOB, the fields involved in the routing of a message can be examined without unpacking and processing the entire message, resulting in low processing time.

As the apps are run directly on the OS but still controlled by the middleware via callbacks, this in term causes slight, but additional overhead. By design, as discussed below, these only occur on starting or stopping the computation, and message passing. Also, a small amount of processing is utilized for application resource monitoring and logging.

The internal structure of the middleware is illustrated in Figure 3. For easy development and maintenance, based on the "separation of responsibilities" design principle, a component-like structure was chosen. Next, the function of each component is summarized.

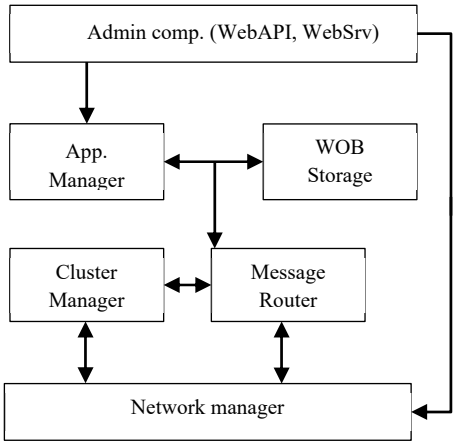

Figure 3: Middleware internal structure.

3.4.1 Network Manager

This component maintains all active peer connections and handles the receiving of incoming messages which then are forwarded to the *Message Router*. Due to overhead considerations, outbound messages are passed by the sender component directly to the *Network Manager* with direct function calls, thus bypassing the *Message Router* entirely.

3.4.2 Message Router

The *Message router* receives the incoming messages from the *Network manager* and routes them to the corresponding component. While the messages are prefixed according to their types, the handling of these overlaps different components.

The message router is also responsible for relaying messages to nodes behind NAT; this routing only involves application-specific messages. In short, if such a message’s *DestinationAddress* is the current node, but the *DestID* differs, then the *DestID* is searched in the cluster members, and if found, the message is relayed directly or via another supervisor (to whom the destination node is connected). If not found, then a delivery failure notification is returned. While this is a solution to messaging nodes behind NAT, an increased number of messages can quickly overload the supervisor.

3.4.3 Cluster Manager

When a node fulfils the supervisor role, then this component is responsible for maintaining all cluster functions, such as: maintaining a local WOB index, answering WOB location queries and storing WOB backups. Synchronization between supervisors is also handled by this component with the use of differential update messages (in other words, only changes are advertised between these nodes).

If a node is a supervisor, then any messages intended to supervisor from the local application is also routed here.

3.4.4 WOB Storage

This component handles the effective storage of WOBs, their backups, and answers the associated request messages. WOB backups are stored at the supervisor level; however, if a node gets disconnected from its cluster, then the WOB backup (created at checkpoint or node shutdown) will be stored on the local peer until the connection is re-established. In such a case, when the node re-enters the network, then a query is made for the current status of the stored WOB, as meanwhile this might have been recovered and under processing or even completed. If this occurs, then the stored WOB in question is discarded from local storage and a new WOB request trigged.

3.4.5 Application Manager

The *App. manager* is responsible for application scheduling (starting, suspending or stopping this), and imposing resource usage limits based on available idle resources and settings. A given application is started before requesting any WOB to ensure that these can start-up and be ready to handle the WOB offerings. As a security measure, all applications are run under a limited OS user.

Application setup is defined in the participating project configuration, which contains the *install*, *run*, and *uninstall* commands.

As the WOB object only contains workload related data, but no executable code and the application itself is downloaded from the project URL, malicious code injection is not possible by a remote node.

3.4.6 Admin Component

Combined with the *Project manager* and the internal *web server*, these are responsible for joining projects, downloading and installing applications, and providing local and possibly remote administration for peer configuration. The design choice was to provide the local admin area via an internal webpage and remote administration via a WebAPI call using an authorization key. This can also be configured to only accept connections from given IP addresses. The local admin portal can also be disabled if remote administration is enabled, which is useful when setting up a local computer network.

3.4.7 Application API

For easing application development for this system, an API is provided that resides before the middleware and "translates" the incoming messages into callbacks, to which the app can register, as illustrated in Figure 4.

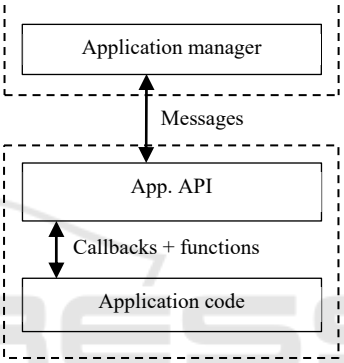

Figure 4: Application API.

The most important *callbacks* and functions can be summarized as:

- The API provided Start() is to be called when the application is ready to receive workload. The request message is dispatched by the API.
- OnWOBInit: called when the workload is received and computing data is to be extracted.
- OnCompute: registered function does the effective computation. Interruption is done via the CanCompute() function, which, if returns false, then the computation is to be suspended. Specialized hardware resources (such as GPU) are to be acquired at the beginning of the registered function. Since there is no guaranty that the computation will be resumed on the current node, the acquired resourced must be released at the end of this function.
- OnCheckpoint: a checkpoint is requested by the middleware. Checkpoint interval is a project setting.
- OnAppExit: WOB data is to be packed as preparation for application exit. The WOB will be transferred off the node (if possible). Even if computing is suspended, this function is called

when a node is shutting down to allow the application to pack the WOB.

- OnWOBRequest: called when another node requests workload from us. Request contains node specifications. The answer is a workload offering. The offered workload is to be locked to prevent multiple offering.
- OnWOBOfferAck: triggered when the requester node accepted our workload offering. The child WOB can be created, prepared and dispatched. On confirmation of receive, the OnWOBChildAck callback is triggered. If the Ack doesn't arrive, then the application can release the workload.
- NotifyWorkloadAvailable() is to be used when additional workload becomes available. This will broadcast a notification message which triggers idle nodes to request workload.
- WOBQuery and WOBLocQuery functions can be used to query the entire WOB with location or only the location of a specific WOB.
- WOBMsg can be used to send application specific messages.
- WOBReqBackup: can be used to retrieve an existing backup of a WOB.
- WOBFinished is to be called when the current WOB is finished. This will be automatically transferred to the parent WOB location, where the OnWOBChildFinished is triggered for its result to be disseminated. The parent acknowledges the WOB with the WOBFinished function, to which this will be removed from the supervisor's indexes and backups.

As mentioned with the presentation of the base model, when a WOB is created, a backup of this is also created to preserve parent-child list consistency.

We can already see that while the middleware offers simplified functions to harness the P2P system's resources, there is additional complexity to application design.

4 TOPOLOGY SIMULATION

Simulation of topology was conducted using a custom, purpose built simulator. The goal was to simulate a variety of user compute resource contribution; hence the number of joining and leaving nodes was generated using an estimated online number of computers per week, obtained from a local internet provider. These numbers differ very little between the weekdays, so one pattern was used for weekday availability and one for the weekend, illustrated in Figure 5.

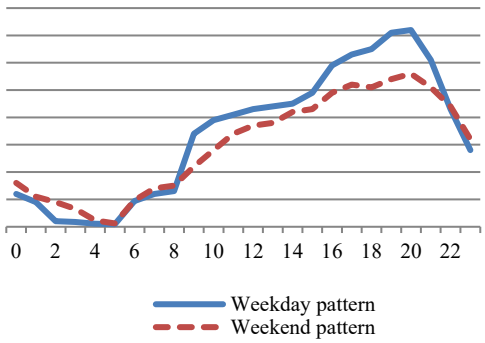

Figure 5: Peer online patterns.

Bandwidth was assumed adequate for the connection capacity. In practice, if such a scenario occurs, the node decreases it's connection capacity accordingly. The simulation was run for two weeks (internal clock) and the data sampled hourly. The parameters are presented in Table 2.

Table 2: Simulation parameters.

Number of nodes	50,000
Node capacity (C_T)	Random from: 50,75,100,125,150
Reserved cap. (C_R)	0.1
Min. supervisors	3
Max. supervisors	0.45 of Min(C_T) Sup.
Cluster split threshold	0.9
Cluster interconnection	2

At the end of the simulation, as illustrated in Figure 6, the following results were obtained: peak number of online nodes: 35995, isolated nodes: 264, total clusters formed 85 and defunct clusters: 79. The first clusters to become defunct were the leaf ones. The other clusters had enough "weight" to survive a large number of disconnecting nodes.

In the best-case scenario, the super-peers managed to organize almost 36000 nodes into 8 clusters with the average size of 4499 (smallest: 4489, largest: 4501) with the longest path of 3 hops. In the worst-case observed, 35995 nodes were organized into 15 clusters with an average size of 2399 (smallest: 2360, largest: 2406) and the longest path of 5 hops between them.

Using a redundant cluster interconnection, no occurrence of isolated clusters was observed during the simulation. Nonetheless, a catastrophic physical network failure can still result in parts of the network becoming isolated.

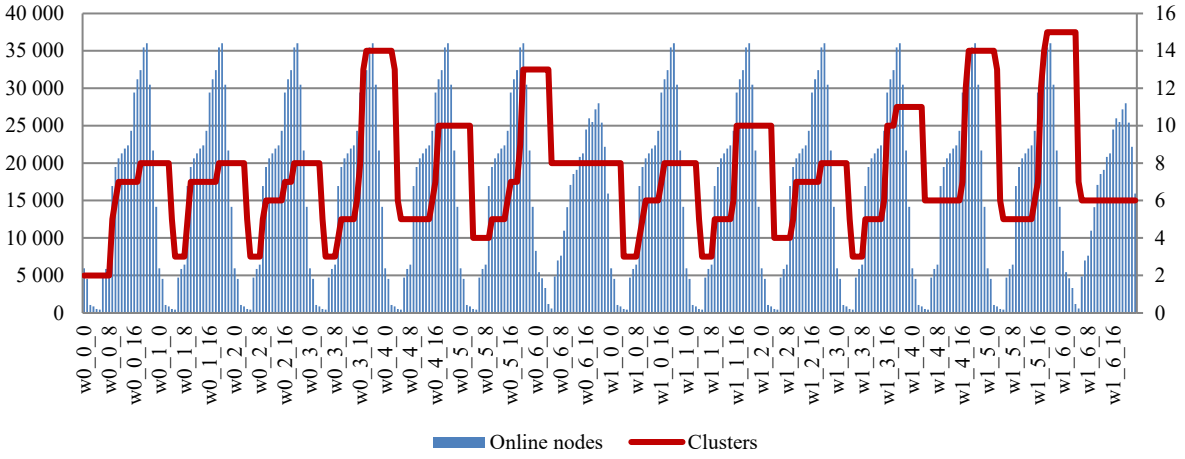

Figure 6: Topology simulation result.

5 CONCLUSIONS

In this paper, a P2P computing system's architecture and design considerations ware presented. Overall, the numerous functions required to operate the system leads to the inevitable increase in both design and implementation complexity.

As in any P2P system design, the topology design is influenced by the operation of the system, specifically, decentralized task coordination, ability to suspend, migrate, recover, and track workloads. Task creation is driven by workload requests, recovery is accomplished using a remote checkpoint system, while workload tracking requires a search mechanism. As mentioned by several literature proposals, super-peers increase the search speed by maintaining a list of connected peer resources, thus reducing the number of peers needed to be queried. In the presented system, super-peers also store the workload checkpoints. Another benefit of clustering is the super-peers ability to limit the workload query messages in certain cases, which otherwise could overload the network. Backup super-peers are used to maintain cluster stability; as one super-peer exit or fails, a backup can take its place, thus keeping the cluster functional. Furthermore, it is desirable to have a reduced number of clusters query messages have to reach. Since each super-peer has a limited number of connections it can accept, further increasing the size of a cluster can be achieved by balancing the cluster members between the super-peers. Concerning the network overlay, the clusters are organized into an extended star topology to reduce the number of hops for a message to take to reach all clusters. Topology simulations to verify the viability of the presented network overlay ware also presented in the paper.

As discussed, such a system design, compared to systems using client-server architecture, comes with an enormous complexity needed to maintain network overlay and provide the specified functionalities. However, this complexity is hidden by the middleware, which provides simple and straightforward functions to utilize and build computing applications. Nonetheless, such applications come with additional complexity. However, the benefit is, as in most P2P systems, the decentralized task coordination, and the unique characteristic of the presented system, the possibility of suspending, transferring and recovering workloads, and also locating workloads allows the possibility of messaging between parallel tasks.

REFERENCES

Anderson, DP. (2004). Boinc: A system for public-resource computing and storage. Proceedings of the 5th IEEE/ACM International Workshop, 4-10. https://doi.org/10.1109/GRID.2004.14

Castella, D., Solsona, F., Gine F. (2015). DisCoP: A P2P Framework for Managing and Searching Computing Markets. Journal of Grid Computing 13, 115-137. https://doi.org/10.1007/s10723-014-9318-3

Chmaj, G. and Walkowiak, K. (2013). A P2P Computing System for Overlay Networks. Future Generation Computer Systems, 29(1), 242-249. https://doi.org/10.1016/j.future.2010.11.009

De, S., Barik, M. S., Banerjee, I. (2016). Goal Based Threat Modeling for Peer-to-Peer Cloud. Procedia Computer

Science (89), 64-72. https://doi.org/10.1016/j.procs.2016.06.010

Filep, L. (2019). Model for Improved Load Balancing in Volunteer Computing Platforms. 15th European, Mediterranean, and Middle Eastern Conference, LNBIP 341, 131–143. https://doi.org/10.1007/978-3-030-11395-7_13

Gailly, J. and Adler, M. (2002), zlib Library. http://zlib.net/index.html

Gomathi, S. and Manimegalai, D. (2013). Hierarchically distributed Peer-to-Peer architecture for computational grid. 2013 International Conference on Green High Performance Computing (ICGHPC). https://doi.org/10.1109/ICGHPC.2013.6533906

Gupta, R., Sekhri, V., Somani, A. (2016). CompuP2P: An architecture for internet computing using Peer-to-Peer networks. IEEE Trans. Parallel Distrib. Syst. 17(11), 1306-1320. https://doi.org/10.1109/TPDS.2006.149

Jesi, G. P., Montresor, A., Babaoglu, O. (2006). Proximity-Aware Superpeer Overlay Topologies. Lecture Notes in Computer Science, 43-57. https://doi.org/10.1007/11767886_4

Lavoie, E., Hendren, L. (2019). Personal volunteer computing. Proceedings of the 16th ACM International Conference on Computing Frontiers (CF '19), 240-246. https://doi.org/10.1145/3310273.3322819

Mitra, B., Ghose, S., Ganguly, N., Peruani, F. (2008). Stability analysis of peer-to-peer networks against churn. Pramana - J Phys, 71-263. https://doi.org/10.1007/s12043-008-0159-0

Pérez-Miguel, C., Miguel-Alonso, J., & Mendiburu, A. (2013). High throughput computing over peer-to-peer networks. Future Generation Computer Systems, 29(1), 352–360. doi: https://doi.org/10.1016/j.future.2011.08.011

Poenaru, A., Istrate, R., Pop, F. (2016). AFT: Adaptive and fault tolerant peer-to-peer overlay - A user-centric solution for data sharing, Future Generation Computer Systems (80), 583-595. http://dx.doi.org/10.1016/j.future.2016.05.022

Qi, X., Qiang, M., Liu, L. (2019). A balanced strategy to improve data invulnerability in structured P2P system. Peer-to-Peer Networking and Applications. https://doi.org/10.1007/s12083-019-00773-9

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S. (2001). A Scalable Content Addressable Network. Proceedings of SIGCOMM 2001, 161-172. https://doi.org/10.1145/383059.383072

Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H. (2001). Chord - A Scalable Peer-to-peer Lookup Service for Internet Applications. Proceedings of SIGCOMM, 149-160. https://doi.org/10.1145/383059.383071

Tiburcio, P.G.S., Spohn, M.A. (2010). Ad hoc Grid: An Adaptive and Self-Organizing Peer-to-Peer Computing Grid. 10th International Conference on Computer and Information Technology (CIT), 225-232. https://doi.org/10.1109/CIT.2010.504

Vimal, S., Srivatsa, S.K. (2019). A file sharing system in peer-to-peer network by a nearness-sensible method. International Journal of Reasoning-based Intelligent Systems (IJRIS), 11 (4). https://dx.doi.org/10.1504/IJRIS.2019.103510

Yang, B., Garcia-Molina, H. (2003). Designing a super-peer network. 19th International Conference on Data Engineering. https://doi.org/doi:10.1109/icde.2003.1260781

Ye, F., Zuo, F., Zhang, S. (2009). Routing Algorithm Based on Gnutella Model. Computational Intelligence and Intelligent Systems. https://doi.org/10.1007/978-3-642-04962-0_2

Zhao, H., Huang, L., Stribling, R., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D. (2004). Tapestry - A Resilient Global-Scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications 22, 41-53. https://doi.org/10.1109/JSAC.2003.818784

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS