Abstract—This paper discuss the design and implementation of an Architecture for Internet–based Distributed Auctions (AIDA). The AIDA system implements responsive and available auction services in a large scale distributed context, such as the Internet, by means of geographically distributed servers. AIDA can support fast auctions, whose duration could be as short as few tents of seconds. Moreover, AIDA does not suffer some drawbacks of the current online auctions, such as the “last minute bidding”. Therefore, AIDA enables types of auction that are rarely implemented online; conversely, these are very common in traditional auctions.

Further characteristics of AIDA are fault tolerance and scalability. The system can tolerate any predefined number of failures, both of servers and of network. The number of servers and the communication bandwidth increase linearly with respect to the number of clients.

I. INTRODUCTION

Internet auctions are gaining increased popularity, and several companies offers this service (e.g., www.eBay.com, auctions.amazon.com, www.yahoo!auction.com, www.ebid.net, to name a few). Some of them offer the auction service on national basis. The number of users, both sellers and buyers, increases constantly. As an example, eBay counts 222 millions of confirmed registered users at the end of the 4-th financial quarter of 2006; these represent an increment of 23% with respect to the same period of the previous year [1].

Commonly, the online auctions last for several days. Typically, a user of these services, a bidder, can submit a bid and, only after an amount of time that can range from hours to days, that user knows whether her/his bid won. This characteristic is a consequence of the asynchronous nature of the Internet, based on a best-effort communication service [2]–[5].

A popular bidders behavior characterizes the most common Internet auctions, i.e., those that ends at a predefined time. The bidders tend to submit their bid as close as possible to the end of the auction, in order to increase the probability that their bid will be the last one, and therefore the winning bid. This phenomenon is called last-minute bidding and has been studied, among others, in [6]. Moreover, the trend is sufficiently popular that both independent software-houses, e.g. [7], and famous auction services, e.g. ebay and Yahoo!Auctions, offer utility programs to automatically submit a new bid by adding a minimum increase to the current winning bid. Those programs submit new bids unless the current winning bid reaches a value previously chosen by the bidder.

A new service is emerging in the auction arena: the “real-time” auctions, such as the “Live Auctions” by eBay. Those new kind of Internet auctions closely resemble the traditional ones. The auction starts at a predefined time, the bidder has to register in advance to participate, and the auctions duration is in the order of few minutes at most. The bidders participate “live” with respect to the auction duration. The possible bidders can browse in advance the “lots” of the auctions.

The actual architectures of the above mentioned services are not public, and we suppose that these rely, in general, on centralized auction server architectures. A centralized architecture cannot deal adequately with issues of service availability and scalability. Typically, such an architecture can be vulnerable to server’s failures, if not equipped with sufficient redundancy; in addition, server’s overloading may occur, if an arbitrary large number of users concurrently access the service. The increasing number of customers of Internet-based auction services suggests that both scalability and availability are crucial issues in the design of those services. In particular, as pointed out in [8], service availability is required as a frequently unavailable service may discourage users from using it, and result in a business loss for its provider.

Owing to the above observations, in this paper we propose an architecture to implement reactive, available, and scalable auctions over the Internet. The architecture, named AIDA (Architecture for Internet–based Distributed Auctions), is scalable with respect to the number of users and can tolerate faults. AIDA is based on services replicated across a number of auction servers distributed over the Internet, and is based on both hierarchical lightweight communications, and coordinator-cohort computation [9].

AIDA offers advantages with respect to both the auction service provider and to the users. Specifically, on the service provider side, our distributed architectures can accommodate an arbitrary number of users by balancing the work load among all the servers implementing the auction service. On the user side, our architecture can support a responsive auction service, provided that the clients be bound to the “most convenient” server (e.g., the most lightly loaded one, the server with the least congested path to the client).

A number of distributed architectures for supporting auction services over the Internet can be found in the literature. In [5]...
the authors propose a hierarchical scalable architecture, while in [8] the servers define a peer group over a wide-area network. In [2] the author points out that scalability and responsiveness are key features for the design of an electronic market server. The soft real-time requirements for an auction server over the Internet are addressed in several works, including [3], [10].

For the sake of conciseness, we will not examine the above mentioned architectures in this paper; the interested reader can refer to the already cited references. Rather, in the rest of this paper, we introduce the fundamental properties of our hierarchy; they are connected exclusively to other servers, i.e. to the plain servers, to the leader, and with each other. In Fig.1 the cohort servers are $S_6, S_7, S_8$. These servers receive the notification of the local best bid from the connected plain servers. Periodically, each cohort server computes the best bid it has received in the last period, if any, and communicates that to the leader. Finally, the cohort servers relay the auction state to the connected plain servers, upon receiving the specific message from the leader.

**Cohort servers:** those servers are at the mid–level of our hierarchy; they are connected exclusively to other servers, i.e. to the plain servers, to the leader, and with each other. In Fig.1 the cohort servers are $S_6, S_7, S_8$. These servers receive the notification of the best bid from the cohort servers. Periodically, each cohort server computes the best bid it has received in the last period, if any, and broadcasts the auction state to the cohort servers. The leader defines the beginning and the ending of the auction. Furthermore, it defines the winning bid at the end of the auction. Possible failures of the leader are managed by the reconfiguration mechanism described later.

The kind of auction defines the data contained in the auction state, e.g. in an English auction the state contains the value of the current higher bid and, probably, a reference to the bidder.

The above description is a simplification, and does not describe the fault tolerance behavior of the system, shown later. Moreover, both the cohorts and the leader might play the role of plain servers, for efficiency reasons; this scenario is ignored in the remaining of the paper for the sake of clarity.

**II. AIDA’S ARCHITECTURE**

**A. Servers hierarchy**

AIDA is an hierarchical, timed, distributed architecture, arranged in three functional levels of abstraction. Each level include a specific class of servers. These servers and their relative levels are described below. The Fig.1 shows an example of the architecture. The nodes labelled $S_1, S_2, S_3, \ldots, S_9$ represent the servers implementing the auction service, and the unlabeled squared nodes represent the clients connected to them. The solid edges in the graph represent communication channels, used to manage the bids, while the dashed edges, fully connecting the set of cohorts servers, are communication channels carrying exclusively coordination messages. The clients have no visibility of each single server, but they perceive the auction system as a coherent whole. Needless to say, the set of servers shown in the figure is largely oversized with respect to the number of clients.

**Plain servers:** these servers are at the lower level in our hierarchy; they accept connections from the clients and are connected to the cohort servers. In Fig.1 the servers $S_1, S_2, S_3, S_4, S_5$ are plain servers. These servers receive clients bids as they are submitted. Periodically, each plain server computes the best bid it has received in the last period, if any, and communicate that to the connected cohort server. A best bid depends on the type of auction being held; e.g. open cry, sealed bid, dutch, double auction. The “best bid” changes over the auction duration, and it will be the winning bid when the auction will terminate. Moreover, the plain servers relay the auction state to the clients, upon receiving the specific messages from their cohort server. Usually, the auction state contains, among the others, the current best bid.

**Leader:** this is a unique server, at the upper level of the tree, connected to the cohorts servers. In Fig.1 the leader is $S_9$. This server receives the notification of the best bid from the cohorts. Periodically, the leader computes the best bid it has received in the last period, if any, and broadcasts the auction state to the cohort servers. The leader defines the beginning and the ending of the auction, furthermore, it defines the winning bid at the end of the auction. Possible failures of the leader are managed by the reconfiguration mechanism described later.

The above description is a simplification, and does not describe the fault tolerance behavior of the system, shown later. Moreover, both the cohorts and the leader might play the role of plain servers, for efficiency reasons; this scenario is ignored in the remaining of the paper for the sake of clarity.

**B. The client program**

The client program is the interface between the bidder and the auction service. The space–time diagram in Fig.2 shows an example of the protocol to join an auction. The labelled solid vertical lines represent the computations of the user, the servers and the Web service. The user is represented by the leftmost line, labelled C. The servers are labelled $S_1, S_2, S_3, S_4, S_5$, and the Web service is labelled with his name. The oblique arrows represent unidirectional messages, and the time flows from top to bottom. The figure represent a client joining the same system depicted in Fig.1.

The users joins the auction by means of a browser request to the Web service of the auction. The Web service answers with a client program, that connects to a server and manage the user participation to the auction. This first step is represented by the first couple of messages in Fig.2, i.e. the “join” and “prog” messages. The client program launches several parallel request of availability to a predefined set of servers, the “available?” message in figure, and waits for the first answer. Then the client program selects the server that sent the first answer it has received, ignoring the successive answers. In Fig.2 the
answers are the “ok” messages, and the client connects to server $S_2$ completing the joining protocol by means of the “subscribe” message. The client program can submit a bid to the chosen server any time it likes, and, almost periodically, it receives the auction state message from the same server. In Fig.2 the client submits a bid and receives an acknowledge, by means of the bid and ack messages; in the meantime the client receives the almost periodic state messages.

The above strategy provides for the selection of the server that is the actual most responsive one. Moreover, this measured responsiveness take into account both the server load and the network congestion. Each client program sticks with the chosen server, unless it detects unsustainable performance, i.e. it cannot keep the user response time below a certain threshold. In the latter case, the client program starts a new selection procedure. Note that the above technique allows for a run–time load balancing with respect to the servers and the network; requiring a negligible effort for the servers, a small amount of work for any client, and a little overhead of the network traffic. Moreover, the presented technique is effective in case of both server and network faults, enabling the client to autonomously connect to another server.

We developed client programs for both desktop and smartphones [11], enabling a multimodal user participation to an auction. Moreover, a bidder could migrate from a device to another while participating to an auction.

III. MAIN AIDA’S PROPERTIES

A. Timelines

We discuss the timelines properties of AIDA by means of the Fig.3, that shows a space–time diagram for a simple auction execution of the system depicted in Fig.1. To simplify the diagram, the figure shows few clients and all the servers. The labelled solid horizontal lines represent the computations of the servers and clients; the time flows from left to right. The small vertical lines on the server’s computations represent time ticks, i.e. when the server computation reaches a tick, it triggers an action accordingly to the server role. The distance between two consecutive ticks is the duration of the server period. The oblique arrows represent messages: the beginning of each arrow is the sender time, while the pointed end represents the receiver time. Note that those times are measured with clocks local to each process, and therefore can solely be assessed by the other servers [12]. We might temporary ignore the points labelled with Greeks letters that are on some server lines, these will be discussed later.

The plain servers send best bid messages to their cohort at a certain pace, such as the cohorts do with respect to the leader. There is no synchronization between the sending periods. The duration of the sending periods is a trade–off between the number of messages, and the responsiveness of the auction system. When a server senses a network congestion it can increase the duration of its period; this leads to both less sent messages to the upper levels, and to less system reactivity. On the other hand, when a server senses an unloaded network it can reduce the duration of its period; this leads to both higher number of messages and better system responsiveness. The servers immediately relay the state messages because there is no convenience into delaying them, and doing that will reduce the system responsiveness.

The Fig.3 shows client $C_1$ submitting the winning bid to its plain server $S_1$, that receives this bid just after a tick. Therefore, $S_1$ will store the received bid until either it receives a better bid, or its current period expires, i.e. its computation reaches the next tick. In the meanwhile, the clients $C_2$ and $C_3$ submit a bid to their plain servers. The server $S_1$ discards the bid from client $C_3$ because this is worse than the local best bid. The server $S_3$ stores the received bid and waits until the next tick to notify its cohort server. Note that the bids of the three clients are independent of each other, because they are submitted in parallel.

As shown in Fig.3, the first plain server that reaches a tick is $S_3$. The notification from $S_3$ to its cohort server $S_2$ arrives just before a tick, then $S_7$ almost immediately notifies the leader $S_9$. In the meanwhile, $S_1$ reaches its tick, and its notification

![Fig. 2. Space–time diagram joining an auction](image)

![Fig. 3. Space-time diagram of some bids to an auction](image)
to $S_0$ arrives just after a tick; again, $S_0$ stores the bid and waits until the next tick to notify the leader.

When the leader reaches its next tick, it multicasts to the cohort servers the actual best bid, i.e. the one previously received from $S_2$. When a cohort receives the multicast from the leader, it immediately relays the message to its connected plain servers. The plain servers do the same, relaying the state message to all of their clients. In the given example, all the clients receive the current best bid message containing the bid from $C_2$.

After the above described multicast, the leader $S_0$ receives the local best bid from cohort server $S_c$. Since this new bid is better than the previous one, the leader stores it until the next tick. As in the previous case, the winning bid finally reaches all the clients after few consecutive multicasts.

B. Responsiveness

We can express the responsiveness $R$ as the time spent by the system to identify the current best bid, and to communicate it to all the clients. In Fig. 3 $R$ is the elapsed time between points $\alpha$ and $\omega$. We can assess the worst case of $R$ by assuming that the maximum latency for any message is $\delta_{max}$. In addition, let the periods of the servers be $\tau_p$, $\tau_c$, and $\tau_l$ for the plains, the cohorts, and the leader respectively.

Summarizing:

$$R \leq \tau_p + \tau_c + \tau_l + 6\delta_{max}$$

We can divide the responsiveness in three sequential phases. The first phase is the time that the leader takes to identify the current best bid, called rise time, that is at most $\tau_p + \tau_c + 3\delta_{max}$. In Fig. 3 the rise time is the elapsed time between points $\alpha$ and $\beta$. The second phase is the leader latency, called coordinator time, that is at most $\tau_l$. In Fig. 3 this is the elapsed time between points $\beta$ and $\lambda$. The third phase is the “diffusion” of the current best bid, called spread time. In Fig. 3 this is the elapsed time between points $\lambda$ and $\omega$.

Note that this assessment of $R$ does not explicitly consider the load condition of the servers, i.e. how long a message has to wait in the queue of the received messages before being evaluated. Such a delay might occur because the server is dealing with messages previously received. According to experimental measurements, we assume that a server can satisfy about 250 requests per second by means of secure communication. In other words, when a server contemporarily receives 250 messages, the last of those messages is processed 1 second after its arrival.

A side effect of the responsiveness is the fairness, $F$, of the system, i.e. the elapsed time between the first and last reception of the same auction state message by the clients. In other words, $F$ represents the advantage of the first client that receives an auction state message with respect to the last client that receives the same message. Fig. 3 shows an example of fairness as the elapsed time between points $\sigma$ and $\omega$. To assess the fairness we consider the maximum difference between the fastest and the slowest spread times for the same auction state message. It could be an acceptable lower-bound to set minimum latency of any message to zero [12], i.e. $\delta_{min} = 0$; therefore

$$F = 3(\delta_{max} - \delta_{min}) \leq 3\delta_{max}$$

C. Scalability

A distributed system is scalable if it remains effective despite of a significant increase in the number of both users and resources [13].

We can assess the amount of resources required by the system with respect to the number of clients. Specifically, we consider both the number of servers and the number of messages exchanged between the servers. The number of plain servers is linearly proportional with respect to the number of clients, and the number of cohort servers is linearly proportional with respect to the number of plain servers. The number of parallel messages exchanged by the servers, is at most twice the number of communication channels between the auction servers.

The performance of AIDA are cpu-bound, i.e. they depend on the processor load and do not require too much bandwidth. While the AIDA servers are dedicated to the service, their load depends on the management of secure communication channels. The size of the messages is invariant with respect to the number of clients and servers, the number of messages is directly proportional to the number of clients.

D. Fault tolerance

The AIDA systems remains available, and properly working, in spite of a predefined number of servers and communications faults. Our system can tolerate the following types of faults:

- channel performance: a communication channel is overloaded, the connection remains active, the messages are delivered with higher latency than expected;
- channel disruption: a communication channel does not delivers messages anymore, it has to be re-established;
- processor performance: a computer is overloaded and responds later than expected;
- processor crash: a computer becomes not working, then all the programs and communications executed by that computer become unavailable. A crashed processor could, possibly, return available after a fresh restart.

Each client regularly receives the auction state message. In regular running the auction state messages is sent almost periodically, its arrival time gives to each client a gauge of the server load plus the network congestion. The missing of an auction state message signals to the client the presence of a fault, either of the server or of the network. In this case the client program could connect to another server, by re-running the procedure previously applied. This operation, named client migration, is completely transparent to the user. This mechanism does not require any effort to the servers, therefore, the crashed ones do not block the migration, i.e. the reconfiguration of the system. Moreover, the migration requires a lightweight work solely to the clients that are migrating, without any kind of coordination with each other.
configuration by a triplet of numbers representing, respectively, the cohort servers, the plain servers, and the clients. Each bar represents the average total time with respect to ten repetitions of each simulation, differentiated by varying the seed of the random generator. The segments at the top of each bar represent the standard deviation of the total time. The bars are composed by three parts, the lower part represents the rise time, the middle part represents the coordination time, and the upper part represents the spread time. Note that the fairness, not shown, is slightly smaller than the spread time, i.e. the system exhibits also a pretty good fairness.

V. CONCLUSION

We implemented a fully working prototype of AIDA, and made a set of experiments that confirmed the expected performances issue. The system is reactive, available, and scalable. Moreover, AIDA does not requires any special hardware or software.

The proposed architecture is well suited to support very fast auctions over the Internet, and can be successfully used to implement uncommon auctions in that environment, such as Dutch and stochastic ending auctions.

REFERENCES