Title: DISTRIBUTED EXECUTION SYSTEM

Abstract: A distributed execution system (DEX System) (100) having a hierarchy (102) in a tree structure, with a central server (104) at the root, hierarchical entities (110) as branches, and canonical entities (108) as leaves. The hierarchical and canonical entities (110, 108) are formed of sets of multiple peer machines (106). The peer machines (106) may be participants in a simulation or may, as hierarchical and canonical entities (110, 108) be executors of the simulation on behalf of the participants. The peer machines (106) in canonical entities (108) execute a region of the simulation in common. The hierarchical entities (110) direct hierarchical and canonical entities (110, 108) lower in the hierarchy (102). Both the hierarchical and canonical entities (110, 108) compare their internal results, and provide votes on those results up the hierarchy (102). To insure secure execution, the peer machines (106) are not permitted to be both participants and executors in the same regions of the simulation.
Published:
— with international search report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
DISTRIBUTED EXECUTION SYSTEM

TECHNICAL FIELD

The present invention relates generally to networked computer systems, and more particularly to a system for distributing and securely executing program applications on such networks. It is anticipated that primary applications of the present invention will include simulations, scientific modeling, multiplayer games, and analyzing large data sets.

BACKGROUND ART

Computer systems are increasingly being networked together to distribute the execution of complex and sizable tasks. For example simulations, scientific modeling, multiplayer games, and the analysis of large data sets (collectively termed all "simulations" herein) may be highly suitable for distributed execution.

In some cases, distributed execution is necessary to accomplish the underlying task. Multiplayer game illustrates this since, to some extent, at least some portion of the game is necessarily executed on game players' personal computers.

In other cases, distributed execution is desirable. For example, consider scientific modeling and the analysis of large data sets. Modeling is notoriously burdensome on the computer systems used. It also tends to have short lived requirements, with systems configured for or powerful enough for executing one model then having to be reconfigured or outright replaced with more powerful computer systems for future models. Rather than using single expensive supercomputers it is therefore preferable to use networked arrangements of many inexpensive computers.

The analysis of large data sets via distributed computing has particularly received recent attention. A widely known example is the SETI@HOME project, wherein participants "donate" unused execution capacity on their personal computers (PCs) to analyze radio telescope data for patterns potentially indicating the presence of extraterrestrial intelligence.

Participants in this project download and install a client software that then communicates with project servers to downloads the data, controls the analysis of that data on the participant's PCs, and reports results back to the project's servers.

Turning now to multiplayer games, which will be used as the primary example of a suitable application for distributed execution here, an emerging trend is the massively
multiplayer online game (MMOG). These require large amounts of execution power, which when concentrated in servers in a conventional client-server scheme become prohibitively expensive. Consequently, only a relatively small number of games are presently available in this compelling genre, despite the significant subscription-based revenue streams that these games can generate if successful. The up-front development costs for providing a MMOG are considerable, and the infrastructure for then just providing the MMOG can cost more than $500,000. Even greater than the up-front costs that these games face, however, is the expense of the ongoing communications bandwidth necessitated by their underlying client-server architecture. A modern based MMOG with 200,000 subscribers can incur as much as $2,000,000 in annual expenses for bandwidth alone.

In simulations generally, as the above game example has particularly illustrated, a large portion of the expenses are tied directly to the centralized client-server systems that have traditionally been relied upon. An attractive way to almost completely eliminate hardware and bandwidth costs would therefore appear to be distributing the simulation execution to participants' computers in a "peer-to-peer" architecture. For instance, in a MMOG using the PCs of the game players to execute the MMOG.

Unfortunately, current peer-to-peer applications suffer from two particular drawbacks that can limit this solution: poor security and poor scalability. Peers on a distributed network, such as the Internet, are generally less secure than central servers, which can be protected behind firewalls and professionally monitored. Also, in existing systems, peer solutions require constant communication among the many nodes (computers) employed, which can strain the resources of a modern network.

Accordingly, what is needed is a solution wherein most aspects of a simulation can be executed on a peer-to-peer network rather than on a central server, yet wherein adequate security and scalability are possible. Such a solution should allow the results from a remotely executed application to be trusted even if performed on an untrusted platform, yet should allow the scope of the application and the number of the participants to grow and shrink as desired.

DISCLOSURE OF INVENTION

Accordingly, it is an object of the present invention to provide a distributed execution system that offers the strengths of both peer-to-peer and client-server architectures without the drawbacks of either.
Briefly, one preferred embodiment of the present invention is a system for distributing execution of a simulation. The system includes a central server and a number of peer machines each able to act as a participant in and an executor of regions of the simulation. One or more hierarchical entities and one or more canonical entities including multiple of these peer machines. The central server, hierarchical entities, and canonical entities are organized to from a hierarchy having a tree structure. The central server is a root, the hierarchical entities are branches, and the canonical entities are leaves. The root is defined as upward, and the leaves are defined as downward. Within each canonical entity, each peer machine executes the same region of the simulation, intercommunicates its result within its canonical entity, performs comparison of all the results of its canonical entity, derives a canonical vote based on its comparison, and communicates that canonical vote up the hierarchy. Within each hierarchical entity, each peer machine derives a present hierarchical vote based on the canonical votes and any other hierarchical votes received from down the hierarchy, and communicates that present hierarchical vote up the hierarchy. The central server assigns the regions to the hierarchical and canonical entities. The central server also assigns the peer machines to be members of the canonical and hierarchical entities, but such that no peer machine is both a participant and an executor with respect to any one of the regions. The central server further directs the hierarchical and canonical entities to execute the regions and the central server integrates the regions into the simulation, thereby distributing execution of the simulation among instances of the peer machines acting as executors to benefit instances of the peer machines acting as participants.

An advantage of the present invention is that it provides a highly secure and scalable system for distributed execution of simulations.

Another advantage of the invention is that it provides levels of performance, reliability, data integrity, and security equivalent to existing client-server systems.

Another advantage of the invention is that it significantly reduces the cost and risk of developing simulations for fickle communities of participants. For instance, subscribing players for massively multiplayer online games (MMOGs) which may not come at all, or which can come in overwhelming numbers.

And another advantage of the invention is that it significantly reduces the costs of operating and supporting simulations.

These and other objects and advantages of the present invention will become clear to those skilled in the art in view of the description of the best presently known mode of carrying out the invention and the industrial applicability of the preferred embodiment as
described herein and as illustrated in the several figures of the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The purposes and advantages of the present invention will be apparent from the following detailed description in conjunction with the appended figures of drawings in which:

FIG. 1 (background art) is a schematic diagram depicting basic elements and concepts of Peer Data Protocol (PDP);

FIG. 2 (background art) is a schematic diagram depicting distributed data storage in PDP;

FIG. 3 is a schematic diagram providing an overview of the inventive DEX system;

FIG. 4 is a block diagram depicting the entities in the DEX system, specifically any of the canonical entity, the hierarchical entity, or the spot-check entity;

FIG. 5 is a block diagram depicting a spot-check entity being used to check on a canonical entity;

FIG. 6 is a block diagram depicting a spot-check entity being used to check on a hierarchical entity;

FIG. 7 is a block diagram depicting how the central server of the DEX system may be a cluster of servers owned and operated by the company conducting a simulation or game;

and

FIG. 8 is a block diagram depicting an example of the use of a subscription in the DEX system.

In the various figures of the drawings, like references are used to denote like or similar elements or steps.

BEST MODE FOR CARRYING OUT THE INVENTION

A preferred embodiment of the present invention is a distributed execution system. As illustrated in the various drawings herein, and particularly in the view of FIG. 3, this preferred embodiment of the invention is depicted by the general reference character 100.

The present inventive DEX system 100 works with another invention by the present inventors, Peer Data Protocol (PDP 10). PDP 10 is the subject of U.S. application No. 848,201, pending, titled "Peer Data Protocol" and by the same inventors, filed on the same date as this application and claiming the benefit of U.S. Provisional Application No. 60/270,821,
filed February 23, 2001), and incorporated herein by reference. The following discussion of
PDP 10 is an introduction of its points that are germane to the DEX system 100.

FIG. 1 (background art) and FIG. 2 (background art) are schematic diagrams
depicting the basic principals of PDP 10. Briefly, a plurality of peer nodes 62 collectively
form a peer-to-peer network 50 wherein data objects 20 owned by owning nodes 16 are
stored on possessing nodes 18. The possessing nodes 18 for this are chosen from among a
larger set of neighbor nodes 56. An owning node 16 can and typically will concurrently be a
neighbor node 56 and possessing node 18 for other peer nodes 62. The most basic principal
of PDP 10 is that ownership and possession of the data objects 20 are separated, and an
owning node 16 can thus be prevented from possessing a data object 20 which it owns. When
an owning node 16 does not possess a data object 20 it follows that it cannot directly perform
operations on that data object 20. This alone provides considerable security for the data
objects 20 in PDP 10.

PDP 10 can, however, optionally employ many other principals, individually or
collectively, to provide additional security. For example, the data object 20 can be encrypted
by both its owning node 16 and the possessing nodes 18. It can also be digitally signed by
these nodes. The data object 20 can also be piece-wise separated and stored across multiple
possessing nodes 18, share-wise stored across multiple possessing nodes 18 (using a sharing
algorithm like Shamir's Secret Sharing Algorithm), or both. The possible permutations of all
of these are numerous, and non germane aspect of this are not discussed here.

The presently preferred embodiment of PDP 10 may separate each data object 20 into
multiple data pieces, particularly if it is large, and copies of the entirety or of the respective
pieces can be made to make multiple share pieces. FIG. 2 depicts a simple example of this. A
given node 14a here provides a data object 20 (FIG. 1) which is to be securely stored. The
data object 20 may be sizable, and thus piece-wise storage is desirable to reduce burdening
the possessing nodes 18. The data object 20 may also be deemed to need increase security,
and piece-wise storage may thus also be desirable for this purpose. Of course, both desires
can also concurrently motivate piece-wise storage. The data object 20 here is therefore stored
as three data pieces 20a-c. Furthermore, data piece 20a is processed to produce data shares
20a-1-3; data piece 20b is processed to produce data shares 20b1-3; and data piece 20c is
processed to produce data shares 20c1-3 (only three shares are shown here due do drawing
capacity limitations, but a larger quantity will typically be used, say, twenty, with three
needed for piece assembly and any greater number usable to verify system integrity). When
the owning node 16 first creates the data pieces 20a-c it can encrypt each piece, digitally sign
each piece, or do both. Similarly, when the possessing nodes 18 receive the data shares 20a1-3, 20b1-3, 20c1-3, they can encrypt each share, digitally sign each share, or do both. The data object 20 is thus stored, in a multiply encrypted and signed form, both piece-wise and share-wise.

This scheme provides strong security against the owning node 16 (also the given node 14a in the example just described for FIG. 2). The owning node 16 cannot directly corrupt the data object 20 since it does not possess it. The owning node 16 would also be hard put to improperly obtain sufficient of the data shares from the different possessing nodes 18 to even attempt assembly of the data object 20. Any such attempt at assembly would also fail unless the encryptions applied by the different respective possessing nodes 18 could be defeated.

This scheme also provides strong security against the possessing nodes 18. An individual possessing node 18 can corrupt its respective data share, but this is readily detectable by comparison of a plurality of the data shares when the respective data pieces are being reassembled for a legal use. A data share deemed to be non-corrupted, by majority vote, can then be used, or this can be treated too highly suspect a situation and a copy of the entire data object 20 that has also been stored in a secure archive can then be used instead.

With respect to the data shares stored on other possessing nodes 18, an individual possessing node 18 has essentially the same standing as a third party, discussed presently. Collectively, the possessing nodes 18 cannot collude to illegally access the data object 20, since the owning node 16 has encrypted it and the possessing nodes 18 presumably do not know what the data shares they store contain. A successful collusion attack would require participation of the owning node 16 and all of the possessing nodes 18, and straightforward schemes can be used for selection of the possessing nodes 18 to make this difficult.

Third parties cannot easily compromise either the security or the integrity of PDP 10.

Lacking all of the data pieces and the multiple encryption keys for them, a third party cannot breach the security of a data object 20. Without the ability to sign as the owning node 16 it cannot mislead the possessing nodes 18. Without the ability to sign as a possessing node 18 it cannot mislead other nodes. Lastly, in the case that any data piece or share is suspected by the nodes 14 to be involved in an impropriety, this can be reported to an authoritative indexing node 68 and the signatures permit easy investigation and hard to controvert accountability.

FIG. 3 is a schematic diagram providing an overview of the DEX system 100. The DEX system 100 includes a hierarchy 102, having one or more central servers 104 at its root and branching outward, ultimately, to user's peer machines 106. Although the similarity between the peer machines 106 here and the peer nodes 62 used to store the data objects 20 in
PDP 10 is strong, these may or may not be equivalent and this should be kept in mind. Distributed execution, verses distributed data storage, is the primary purpose of the DEX system 100.

Rather than having one conventional central server system simulate everything simultaneously, in the DEX system 100 a simulation is broken into portions and each portion is simulated by independent groups of the peer machines 106, called canonical entities 108. These canonical entities 108 are, in turn, governed by other groups of peer machines 106 called hierarchical entities 110, for management and particularly for communication up and down the hierarchy 102. Optionally, groups of the peer machines 106 may also be members of groups called spot-check entities 112.

The DEX system 100 is particularly able to execute multi participant simulations where preserving data security and integrity are important, despite individual participant efforts to undermine these goals and even despite a considerable amount of multi participant collusion to do this. The example which will be used herein is the massively multiplayer online game (MMOG), although numerous examples drawn from other genres can also employ the inventive DEX system 100.

In simulations, generally, and particularly in the context of MMOG's a geographical analogy is useful. The simulated entirety may be viewed as the "world," and each piece being simulated as a "region." A participant (or "player" in a game) may thus be "present" in one region and be effected by or have an effect on one or more other regions. These can be termed "local" regions. No participant, however, has worldwide presence or effect, and the other regions of the world thus are "non-local" regions. It follows that some such non-local regions are more distant from a participant's local regions than are others.

This analogy breaks down in one particular regard, but in a highly useful way. The world being simulated is virtual and executing it must take place somewhere. In a conventional scheme this is on a central server, with all of the burdens which that entails. In contrast, in the DEX system 100 is done by distributing the execution of the regions to participants having no interest in those regions. That is, no peer machine 106 is permitted to be a member of canonical entity 108, a hierarchical entity 110, or a spot-check entity 112 handling a region in which that peer machine 106 has an interest.

Currently, most secure networked systems use client-server architecture. Systems that have scaled well, such as domain name services (DNS) and e-mail, rely on a hierarchy for their scaling properties. The most uniform distribution of load on a system's component nodes is generally obtained with a peer-to-peer architecture. In the DEX system 100 the strengths of
all three types of systems (client-server, hierarchical, and peer-to-peer) are capitalized on and the weaknesses are avoided.

The DEX system 100 provides all of the networking, security, persistence and scalability characteristics needed for a large scale multi participant interactive simulation, e.g., a MMOG. Moreover, the DEX system 100 presents simulation and game developers with a scheme that is fundamentally very similar to the client-server model with which they are familiar. Thus, porting an existing client-server simulation or game to the DEX system 100 is straightforward and developing new simulations and games is greatly simplified.

Because DEX system 100 is a hybrid system, it functions much like a traditional client-server system, except that instead of handling execution on centralized servers the DEX system 100 distributes this over many peer servers. These peer servers are simply user's peer machines 106, that execute tasks for other peer machines 106.

This basic relationship is organized to create three core systems. A client-server system is executed between each client node (a given peer machine 106) and a group of other peer machines 106. The hierarchy 102 is then constructed for managing, coordinating, and enabling communication among these groups of peer machines 106. And the peer-to-peer canonical entities 108 are formed among the peer machines 106 in each group for monitoring the reliability and security of the DEX system 100.

FIG. 4 is a block diagram depicting any of the entities in the DEX system 100, i.e., any of the canonical entity 108, the hierarchical entity 110, or the spot-check entity 112. All of these entities are formed of multiple peer machines 106. In FIG. 4, five peer machines 106 are shown but this is a matter of design choice and not a requirement. The peer machines 106 within an entity execute the same tasks, intercommunicate their results (as stylistically depicted with dashed lines here), compare those results and "vote." It should particularly be noted that the voting scheme used here is in the nature of a council voting. All members of the council vote and are aware of the votes of all other members, yet each member independently reports the results of the vote to the entities interested in its outcome.

Most typically, the vote will be unanimous and all of the peer machines 106 in the canonical entity 108 will report this to the entity's parent in the hierarchy 102 and carry on. Occasionally the vote will one short of unanimous, or one peer machine 106 may fail to participate. Unless the task of the canonical entity 108 is deemed particularly sensitive, which is a valid option, the peer machines 106 will report this and the DEX system 100 will also carry on. Any lesser vote, say, even a simple majority like three of five here, is a less likely scenario and usually will merit administrative action from a parent entity. A key point to be
taken here, however, is that the peer machines 106 in the entities work in parallel.

FIG. 5 is a block diagram depicting a spot-check entity 112 being used to check on a canonical entity 108, such as the case represented at section 114 in FIG. 3. The flow of information (messages, instructions, data, results, etc.) here is depicted by arrowed lines 116, and all of these are arrowed at both ends here to represent that the parent hierarchical entity 110 provides both the child canonical entity 108 and the child spot-check entity 112 here with the same information, and that it receives back information from both. Whether the information received back is the same is, of course, another matter.

FIG. 6 is a block diagram depicting a spot-check entity 112 being used to check on a hierarchical entity 110, such as the case represented at section 118 in FIG. 3. Arrowed lines 116 are again used to depict the flow of information. Here two canonical entities 108 are children of a first hierarchical entity 110, that is itself a child of a second hierarchical entity 110. The second, superior hierarchical entity 110 provides information to the first, lower hierarchical entity 110. The second hierarchical entity 110 may concurrently provide the same information to the spot-check entity 112, but this need not always be the case. The first, lower hierarchical entity 110 provides information to the two canonical entities 108. The peer machines 106 in each canonical entity 108 then perform tasks as required, intercommunicate their results, vote, and communicate information to both the first hierarchical entity 110 and to the spot-check entity 112. The peer machines 106 in the first hierarchical entity 110 then perform tasks as required, intercommunicate their results, vote, and communicate information to the second, superior hierarchical entity 110. Essentially simultaneously, the peer machines 106 in the spot-check entity 112 perform the same tasks, intercommunicate their results, vote, and communicate information to the second hierarchical entity 110. The peer machines 106 in the second hierarchical entity 110 can then compare the results from both the first hierarchical entity 110 and the spot-check entity 112, and initiate appropriate action if there is any discrepancy.

Before concluding with FIG. 6, it should be noted that the arrowed lines 116 depicting the flow of information here are not all arrowed at both ends. This depicts the general case that the spot-check entity 112 will not communicate information to the canonical entities 108.

The role of the spot-check entity 112 is to check on the actions of the first hierarchical entity 110 and report its findings to the second hierarchical entity 110.

Let's consider the geographic analogy for simulations again, but now specifically for computer games. Because the processing of most games is essentially a mathematical simulation that is based on variables input by their players, distributing the execution of
games requires that the game process be broken into many relatively independent parts. Conveniently, most multiplayer games are built on the premise that an "avatar" at a specific location in the game world represents each player. By dividing the game world geographically, the game process can be easily separated into the relatively independent parts necessary for distributed execution.

In the DEX system 100 distributes execution across its hierarchy 102 of hierarchical entities 110 and canonical entities 108 of user's peer machines 106. The central server 104, present at the root of the hierarchy 102, is not generally used for game execution, as is the conventional approach. Instead, in the DEX system 100 the results of the pieces of the game or simulation are transferred among the canonical entities 108 only as needed. Over time, the entire game world is "backed up" as the game state that is managed by each canonical entity 108 slowly migrates up the hierarchy 102 to the central server 104.

Of course, the members of these canonical entities 108 are the peer machines 106, operated by players who are playing the game. So, while a player is involved in the game experience, his or her computer (peer machine 106) may simultaneously be acting as a member of a canonical entity 108 and simulating someone else's game experience, or as a member of a hierarchical entity 110 and managing the DEX system 100, or performing some other task. These tasks and the role each peer machine 106 plays in them are outlined below.

The canonical entity 108 is a small group of peer machines 106 (arbitrarily, one can assume five for this discussion) that are responsible for a subset of the total game simulation. These peer machines 106 maintain synchronization among themselves using a voting technique, described presently. The DEX system 100 is designed for applications in which each user's peer machine 106 has a vested interest only in the results from a subset of the total simulation (i.e., events taking place relatively near that player). Results that interest a given peer machine 106 are considered to be "local" to it.

To prevent these peer machines 106 from tampering with the game, a peer machine 106 in a canonical entity 108 is only assigned responsibility for simulating parts of the game world that are distant from where it is actually playing, itself. In other words, no peer machine 106 is permitted to perform executions on data local to itself. Thus, no member of a canonical entity 108 has a vested interest in tampering with the results of the game for which it is responsible.

As mentioned above, the peer machines 106 of the canonical entity 108 maintain a synchronized execution of their assigned part of the game world. At each step, these peer machines 106 communicate with each other, verifying that all the peer machines 106 have
obtained the same results. Because the members of the canonical entity 108 are required to vote amongst themselves regarding the state of the simulation that they are executing, if one of the peer machines 106 has been corrupted, the others can overrule it and update it with their results. In the highly unlikely case that more than one peer machine 106 disagrees, the canonical entity 108 can request arbitration from their parent in the hierarchy 102 (see below).

By multithreading the voting, execution, and input systems for the peer machines 106 in a canonical entity 108, the DEX system 100 ensures that the gameplay response is rapid and that the player experience is consistent. If one of the peer machines 106 in the canonical entity 108 computes an incorrect result or unexpectedly disconnects from the DEX system 100, the players in that region of the world will experience at worst a momentary lag before another member of the canonical entity 108 takes over the tasks of the departed peer machine 106.

The peer machines 106 serving as the canonical entity 108 for a specific area of the game world are randomly assigned and may be periodically shuffled, making it extremely unlikely that multiple members of a canonical entity 108 will have the time and opportunity to collude with each other in order to corrupt the DEX system 100. Furthermore, the synchronization system used permits easy detection of most intrusions, providing additional security and quick responsiveness to breaches.

The results of the simulation are communicated to two different groups. First, each step of the execution is communicated to the players (peer machines 106) for whom a remote (non-local) canonical entity 108 is acting as a server. Thus, the dynamic of the game is fundamentally no different than that of current client-server systems. The executing peer machines 106 send their actions "up" to the canonical entity 108 ("server"). The results of the actions are processed by the canonical entity 108 and then immediately sent back "down" to the player's peer machines 106, as well as typically being sent up the hierarchy 102 to one or more parent hierarchical entities 110.

In addition to maintaining this familiar client-server dynamic, the members of the canonical entity 108 transmit their results to each other, in order to vote, and to their parents in the hierarchy 102 (see below).

By sending the results of the game simulation up to the members of the hierarchy 102 above them, the canonical entity 108 creates three important results. First, the overall game state is updated through this upward flow of results through the hierarchy 102. Second, the parents of a canonical entity 108 may perform spot checks to verify that the entire canonical
entity 108 is executing the game properly (a counter-measure against collusion). Finally, it provides the parents of the canonical entity 108 with the recent execution results they might need in order to successfully arbitrate in case the canonical entity 108 is unable to reach a majority decision regarding the result of the simulation.

If the game designers believe that there are likely to be a large number of attempted security violations, they can employ another counter-measure against collusion by malicious users by periodically rotating or shuffling the members of each canonical entity 108. Though the data needed to run a part of the simulation is larger than the execution results, it is generally small enough to transfer quickly. Thus, it is possible to swap peer machines 106 around the DEX system 100 so that no peer machine 106 is a member of a canonical entity 108 responsible for any given area of the game world for very long, without significantly affecting the user experience. Doing so dramatically increases the difficulty of coordinating an attack on the DEX system 100.

Coordination between the canonical entity 108 and the hierarchy 102 allows monitoring of the DEX system 100 that is as or more effective than that available in a traditional client-server solution. By comparing the peer machines 106 in a canonical entity 108 (or a hierarchical entity 110), it is easy to quickly identify potential illegal activities. For example, if a peer machine 106 in a canonical entity 108 repeatedly accesses data that its peer machines 106 do not, it can be easily detected as a probable information-based cheat.

Alternately, if a single peer machine 106 continually disagrees with its peer machines 106 in an entity, or if all regularly disagree, the DEX system 100 may interpret that as an intrusion, move that part of the simulation elsewhere, and flag the offending peer machines 106 for additional oversight or administrative action.

To ensure the integrity of persistent data and game execution, as well as to provide a consistent level of game experience and to obscure the partitioning of the game world, the DEX system 100 maintains a high level of synchronization amongst the individual segments of the world. Client-server systems accomplish this by simply having the entire world be simulated on a single server or a small set of fixed servers that are synchronized on an internal LAN. The DEX system 100, however, must maintain this level of synchronization on a broad, dynamic peer network (i.e., across the hierarchy 102).

To accomplish this overall world synchronization and hence maintain a consistency of gameplay experience and persistent data, the DEX system 100 makes use of two synchronization mechanisms. First, the individual peer machines 106 of each canonical entity 108 maintain internal synchronization with each other to ensure agreement on the results of
the execution in its region. Second, each region of the game world is simulated by canonical entities 108 that maintain synchronization with each other and with the central server 104.

The overall game world synchronization depends on the individual peer machines 106 that compose each canonical entity 108 being internally synchronized with each other. This "Internal Entity" synchronization is an inherent part of the voting process that the canonical entities 108 use to ensure that their component peer machines 106 agree with each other on the results of the game simulation.

When a canonical entity 108 is first formed, each peer machine 106 in it synchronizes its game clock with a game time server in the central server 104 (which is owned and maintained by the game's publisher) using Network Time Protocol (NTP), which tells the peer machine 106 the synchronized time and a window of precision. By measuring this window of precision (and re-synchronizing if it is unacceptable), it is possible to confirm that each peer machine 106 is synchronized at the beginning of the entity formation and voting process.

The peer machines 106 in the canonical entity 108 establish encrypted communications channels with each other and begin executing the simulation for which they are responsible. At regular intervals, the results of the simulation are hashed into a digital signature block and transmitted to each other peer machine 106 in the canonical entity 108. Each peer machine 106 can then compare its result with the results of its peers in the canonical entity 108 to confirm the outcome of the simulation for that "step." This process is called voting.

Because this voting process has an unavoidable latency associated with each transmission of data, it occurs asynchronously with the execution of the game world itself. Thus, say, each 0.2 seconds of game time, each peer machine 106 stops executing the game world, calculates the result of its vote, transmits that result, and starts executing the game world again. At a future time (when all the votes have been received), it compares the results of the votes and takes the appropriate action. If one of the peer machines 106 in the canonical entity 108 has voted incorrectly, the remaining peer machines 106 update that peer machine 106 with the new world state and they all continue. If more than one peer machine 106 has voted incorrectly, the canonical entity 108 informs its managing hierarchical entity 110, which deals with the problem.

To ensure that the result of the vote will be identical when the game clocks are received, all communications in the DEX system 100 may be time-stamped with the game time that they are initiated. The peer machines 106 in the canonical entity 108 are then able to
process commands based on when they were sent, not when they were received. Thus, if all
clocks in a given canonical entity 108 are synchronized, the contents of the command queue
held by each peer machine 106 in the canonical entity 108 will be identical at the moment of
voting.

Just as in global synchronization of the overall game world, the internal
synchronization state of a given canonical entity 108 is subject to disruption from variances
in system clocks and overloaded processors. As such, periodic checks must be performed to
ensure that the peer machines 106 in a given canonical entity 108 are still in synch with each
other. The bounds of error on internal votes are substantially tighter than those for
synchronization of the overall game state, and the acceptable limits of precision on the
internal synchronization are correspondingly higher. This requires that internal
synchronization checks must occur much more frequently.

Fortunately, the voting mechanism itself provides the solution to this dilemma. When
each vote is received, the time of receipt is recorded and compared to the predicted latency
between the two peer machines 106 in the canonical entity 108 and the time that the receiving
peer machine 106 sent out its vote. This comparison can allow the receiving peer machine
106 to determine the likelihood that the sending peer machine 106 sent its vote at the same
time as the receiving peer machine 106, or before or after. Corrective action can then be
taken.

By basing the voting scheme of canonical entities 108 on a regular game time event,
and stopping execution of the game world for the voting step, the DEX system 100 can be
allowed to function with any existing or proposed game engine synchronization method
today. In effect, the game engine used to simulate the game world in each region is
independent of the voting system. As long as the voting system can interrupt the game engine
regularly, and has a designated data structure representing the results of the simulation,
voting and synchronization in the DEX system 100 will function.

Overall game world synchronization is fairly simple to achieve once each canonical
entity 108 is synchronized internally. When each peer machine 106 in the game logs into the
DEX system 100, it synchronizes its game clock with the game time server using NTP. Once
a given canonical entity 108 has achieved internal synchronization, it resynchronizes with the
game time server to ensure that it has not lagged during its internal synchronization process,
and then begins simulating the game.

If the DEX system 100 were running on controlled hardware or system clocks on
player's peer machines 106 could be trusted, this would be sufficient to maintain game time
synchronization. Unfortunately, this is not the case. The DEX system 100 therefore should
correct for system clocks and timers which drift unpredictably, or for individual peer
machines 106 becoming overloaded and falling behind.

Fortunately, it is not necessary to maintain synchronized "real" time. Instead, the DEX
system 100 can maintain a synchronous game time. The term "game time" refers to an
arbitrary clock correlated to the passage of events in the game. Any two canonical entities
108 in the DEX system 100 should agree on the current game time, but are not necessarily
required to agree on a current real time.

To accomplish this game time synchronization, each canonical entity 108
resynchronizes with the game time server (in the central server 104) periodically to
compensate for drift. If the current game time of the canonical entity 108 is ahead of the
game time server, then the canonical entity 108 is running too fast, and needs to slow down
slightly. This can be accomplished by spending slightly fewer processor cycles running the
game, or by assigning more of the game world to that canonical entity 108. If the current
game time of the canonical entity 108 behind the game time server, the canonical entity 108
is running too slow, and needs to speed up. This can be accomplished by moving some of that
region of the game world for that canonical entity 108 to another canonical entity 108, thus
relieving load. If the current game time of the canonical entity 108 is within an acceptable
margin of the game time server, then no action needs to be taken. Of course, "repeat
offenders" can be replaced, reassigned, or otherwise dealt with to maintain consistent game
play.

The hierarchy 102 allows the DEX system 100 to maintain accountability and
 oversight of the simulation execution despite its distribution across network of peer machines
106. It is responsible for overseeing and coordinating the activities of the canonical entities
108 that perform the actual execution, as described above.

In a MMOG there will be many canonical entities 108 at a given time (one for each
region of the game world being actively simulated). Players in these games will need to be
able to move across the game world, passing through areas being simulated by different
canonical entities 108. It will be necessary to make that transitioning process transparent to
the player. Additionally, in most games, the complexity of the simulation increases when
more players congregate in a smaller area. As a result, the game world will need to be broken
down into smaller parts in areas like "cities," where more players' avatars are congregated, to
avoid excessive load on the canonical entities 108 acting as execution servers. In situations
such as these it is crucial to manage the canonical entities 108 so that players (and events that
affect the players) can smoothly pass between regions of the game world governed by the different canonical entities 108.

The hierarchical entity 110 is a group of peer machines 106 assigned to provide this coordination. To protect the DEX system 100 against malicious activities by individual users, the hierarchical entities 110 are composed of multiple synchronized peer machines 106 (say, five like the canonical entities 108) and use the same voting mechanisms to stay synchronized and resolve disputes as do the canonical entities 108.

Integrating larger numbers of players into the game is accomplished by including these hierarchical entities 110 in the hierarchy 102, or tree. If there are more canonical entities 108 in an area than the assigned hierarchical entity 110 can handle, then additional hierarchical entities 110 will be assigned. Each hierarchical entity 110 coordinates the efforts of its immediate children and works with its parent and siblings to integrate results up the hierarchy 102. The hierarchy 102 then naturally expands to accommodate increasing numbers of players and hence canonical entities 108. The root of the hierarchy 102 is the central server 104, which typically is a small set of server systems owned and managed by the game company and that reside behind its corporate firewall.

The primary responsibility of a hierarchical entity 110 is management of the entities beneath it. Because hierarchical entities 110 do not simulate the game world, they have the capacity to monitor the resource usage of their children and themselves, and to modify the structure of the hierarchy 102 such that no entity is over- or under-loaded. The hierarchical entities 110 are also responsible for ensuring that each canonical entity 108 is simulating the game correctly and accurately, and preventing cheating or other attacks against the DEX system 100.

The main method used by the hierarchical entities 110 to meet these responsibilities is the creation and dissolution of other entities (child canonical entities 108, other hierarchical entities 110, and spot-check entities 112). The hierarchical entities 110 are permitted to insert new entities into the hierarchy 102 if their children's resource usage grows too high -- reducing the child's load by diverting a portion of its responsibilities to the new entity -- or to destroy underutilized entities by transferring their duties to other entities.

If a hierarchical entity 110 suspects that one of the entities which reports to it is not performing its duties adequately, it may insert a spot-check entity 112 into the hierarchy 102 as a check of the existing entity. This spot-check entity 112 will perform the same actions as the existing entity, but will report its results to the hierarchical entity 110. The hierarchical entity 110 will then be able to compare the results of the spot-check entity 112 with those of
the existing entity.

The canonical entities 108 simulating adjacent areas of the game world should be placed as "close" to each other in the hierarchy 102 as possible (ideally, under the same hierarchical entity 110). This maximizes the efficiency of the hierarchy 102. Additionally, it creates an important benefit because canonical entities 108 are selected by ensuring that the game world distance between the peer machines 106 and any part of the simulation in which they may hold an interest always exceeds a certain fixed value. Thus, if a peer machine 106 in an entity has a vested interest in a given part of the game world, the DEX system 100 can guarantee that such a peer machine 106 will be performing executions only on parts of the game world that are far away from where its user is located, and consequently that user will be unlikely to be able to have any influence on their own role in the game.

As the number of players in the game world shrinks (say, as players log off for the day), the hierarchy 102 slowly shrinks, reducing the number of canonical entities 108. Areas of the game world that are sparsely populated may now be rotated onto the central server 104 for execution and storage.

The DEX system 100 offers two options for persistent data backup and world state maintenance. Under the centralized data backup scheme, data is backed up on the central server 104 at all times; under the decentralized scheme, data is only stored on the central server 104 when no peer machines 106 are available to carry the load.

First lets consider centralized data backup. At each step in the hierarchy 102, the entities periodically update their parents with the results of the simulation being executed by the canonical entities 108 beneath them. Over time, the results of the execution migrate up the tree of the hierarchy 102 to the root central server 104 that holds the entire game world. This allows the results of the entire problem to be assembled at a single point while minimizing the amount of redundant or inconsistent data that arrives at the central server 104.

With centralized data backup, game developers can monitor the game world at any time, searching for inconsistencies. They are also assured of complete game world data "snapshots" being stored at various moments in the past, say, 1 second, 10 seconds, 100 seconds, and 1000 seconds, in addition to the normal tape backups. This allows the game developers to perform rollbacks or restores with far less disruption to the game than restoring from tape backups that are 24 or 48 hours old. The disadvantage of centralized data backup is that it does require a small amount of bandwidth on the central server 104, and that it places a higher load on the hierarchical entities 110.

Now lets consider decentralized data backup. For game developers who are concerned
about reducing their server bandwidth even further, the DEX system 100 offers a system for
decentralized persistence. Under this option, the canonical entities 108 back up their game
world data to a hierarchical entity 110, but this data is not further transmitted up the hierarchy
102. Instead, each successive backup overwrites the previous backup.

To provide additional protection against attacks against canonical entities 108, each
canonical entity 108 in the decentralized data backup system may be "shadowed" by another
canonical entity 108. Under this system, the official canonical entity 108 will forward its
inputs (the player commands) to the shadow canonical entity 108, which will perform the
same execution. Thus, if the original canonical entity 108 is attacked and compromised, the
results of the simulation are still available on the shadow canonical entity 108. Essentially,
this is a continuous "spot check" process and provides a level of security above and beyond
the normal spot check available to the hierarchical entity 110.

Under the decentralized backup system, the game world state is only transferred onto
the central server 104 when the number of players in a region drops low enough that each
region is moved off its canonical entity 108 and that entity is disbanded. This will occur as
players log out of the game. The obvious advantage to this is dramatically reduced bandwidth
on the central server 104, even compared to the extremely low bandwidth costs of the
centralized backup system above. It is noteworthy, however, that this backup option renders
the game world vulnerable to large-scale internet failures; if a substantial percentage
(approximately 40%) of the peer machines 106 are disconnected at the same time, all
simulation results since the last low-activity period will be lost.

In addition to the persistent data storage and backup outlined above, certain
transactions may be designated as "secure transactions," requiring more monitoring and
protection against cheating than most game actions. These transactions may be performed
directly at the central server 104 and integrated into the game world.

The root of the hierarchy 102 is the central server 104. FIG. 7 is a block diagram
depicting how the central server 104 may be a cluster of servers owned and operated by the
company conducting the simulation or game. These servers are responsible for backing up
game data, performing secure transactions, monitoring the hierarchy 102 itself, and handling
player login and authentication.

When players log into the DEX system 100 to begin a game session, their usernames
and passwords may be verified by an authentication service 120, which interfaces with a
billing service 122 to ensure that the player's account is up to date. The player's peer machine
106 is transferred to a machine management service 124, which directs a tree management
service 126 to manage the available peer resources in the DEX system 100 and which directs a patch service 128 to update the player's peer machine 106 with any patches that have become available since the player last logged in.

Communications between the servers of the central server 104 and with the peer machines 106 may be handled by an interface service 130, which can also be responsible for encryption and decryption. The interface service 130 generates a unique session key each time the player logs in, ensuring secure communications.

The tree management service 126 may be responsible for ensuring the smooth functioning of the hierarchy 102, and managing its top levels. In conjunction with the machine management service 124, it can identify the peer machines 106 which the hierarchical entities 110 use to create new entities, and it provide a known third party for creating secure encrypted communications channels between the user's peer machines 106 in the DEX system 100.

A persistent data service 132 can be provided and responsible for maintaining a most current database of the game world state possible. This service or server receives backups from the centralized backup system, as well as the results of secure transactions and any simulation results from regions being executed on a game execution service 134 (see below). The persistent data service 132 allows IT staff to monitor the quality of gameplay.

Finally, the game execution service 134 can be provided and simulate any areas of the game that are not being simulated on canonical entities 108, as well as performing secure transactions. This server can provide a known hardware platform for absolutely secure execution of the game, or for computationally intensive simulation. Regions of the world that overload the resources of any canonical entity 108 may be transferred to this game execution service 134, which has no such limitations.

The multiple components of the DEX system 100 may be combined to fit any game's requirements. Game developers may choose to designate part of their simulation as secured transactions, part as centralized backup data, and part as decentralized backup data to meet their own requirements for server bandwidth load and security. The conditions under which the hierarchy 102 transfers execution to the game execution service 134 may be set to balance security, speed of response, and cost. Patches may be associated with billing cycles, game status, or other data. The flexibility of the DEX system 100 ensures that it will function and meet nearly any set of requirements for any MMOG.

Because the game simulation is run on peer machines 106 rather than servers, the size of the game segment that these peer machines 106 are capable of simulating is limited. In
order to make sure that these peer machines 106 are not over-burdened, the number of players present in any single region of the game being simulated by a canonical entity 108 can be limited based upon the bandwidth and processor resources available within the canonical entity 108. The result of this limitation is that communications between canonical entities 108 will occur frequently, and must be accommodated by the DEX system 100. For example, a player may choose to look into an adjacent area, or may perform a game action on their peer machine 106 that has an affect outside of the area being simulated by its governing canonical entity 108.

In order to facilitate these communications between canonical entities 108, each entity maintains a cache of "nearby" entities and their corresponding regions. When the canonical entities 108 are rotated, changed, or transferred to different peer machines 106, the addresses in this cache are automatically updated. This allows the canonical entity 108 to immediately lookup the address of a nearby region in order to communicate with it. Events that are to be sent to a distant region require the canonical entity 108 to query the central server 104, which slightly increases the lag of such an event. Fortunately, the caching system here makes this type of query infrequent.

In order to ensure that the information flow between the canonical entities 108 occurs smoothly, communications between the canonical entities 108 in the DEX system 100 are governed by three rules. First, game events that are initiated in a particular part of the game world are always created by the canonical entity 108 responsible for that part of the game world. Second, game events that will affect a part of the game world outside of the area in which they were initiated are always passed directly to the affected canonical entity 108. Third, all game events are classified either as actions (events that will have an effect on the game world) or subscriptions (events that request information from other canonical entities 108).

When a player takes an action that may have an effect in another part of the game world, the canonical entity 108 responsible for that player initiates an event and passes it directly to the canonical entity 108 responsible for the affected segment of the game world. The initiating canonical entity 108 then returns to "business as usual," because it is not responsible for any effects of that action outside of its own region. The event received by the second canonical entity 108 is simply inserted into its request queue and processed during the next simulation frame, along with the user inputs from that region.

Subscriptions are handled just like actions, with the following addition. If a player requires information not possessed by that player's canonical entity 108, the entity sends a
subscription request event to the appropriate canonical entity 108, which deals with it exactly as it would deal with a normal action. The target canonical entity 108 begins transmitting update information about the specified area of its region to the player's canonical entity 108. The player's canonical entity 108 then transmits whatever information is appropriate from this data to the player's peer machine 106, enabling him to "see" the adjacent area. When the player moves away from the border between these areas, or no longer needs access to this information, the canonical entity 108 may cancel the subscription by sending a cancel subscription event message, releasing the target canonical entity 108 from the responsibility of sending the game information each frame.

FIG. 8 is a block diagram depicting an example of a subscription in the DEX system 100. Imagine that a player in an MMOG chooses to shoot an arrow at an opponent that he can see in the distance who happens to be in an adjacent region of the game being governed by an adjacent canonical entity 108. Before the player chooses to fire the arrow, he would need to be able to see the opponent in the distance (in the adjacent region of the game world). When the player first turned to view the area containing the opponent, the player's canonical entity 108 would send a vision subscription event for the area to the adjacent canonical entity 108. On each successive game frame, the adjacent canonical entity 108 would send a vision update to the player's canonical entity 108, which then passes the information to the player when he looks in that direction.

When the player takes whatever actions are necessary to fire his arrow at his opponent, his canonical entity 108 creates the game effect, "flying arrow," and passes it to the adjacent canonical entity 108 to be resolved. After the "flying arrow" event leaves the region of the first canonical entity 108, that entity returns to its normal tasks. The adjacent canonical entity 108 accepts the "flying arrow" event, determines that it has struck the opponent, and continues with its simulation tasks, which now include the effects of the "flying arrow."

Finally, now that the player's arrow has struck the opponent, the adjacent canonical entity 108 passes the results of this strike to the player's canonical entity 108, as part of the ongoing vision subscription, allowing the player to see his arrow strike his opponent. As usual, both canonical entities 108 update their relevant governing hierarchical entity 110 each simulation frame.

Persistent-state online games often include data objects and virtual items that possess real-world value. As a result, such games require that transactions regarding these data items be monitored and that additional precautions be taken to prevent data duplication, modification, or theft. The DEX system 100 addresses these concerns by implementing a
secure transaction system that offers the game developer the option of running the most critical, secure, parts of the game on the protected environment of the central server 104.

Any class of transaction that occurs in the game simulation can be specified as a secure transaction to the DEX system 100. Unlike normal execution, these secure transactions are then routed through the secure central server 104, which reside behind the corporate firewall.

With reference again to FIG. 7, secure data objects 136 (not to be confused with the data objects 20 in the prior art PDP 10, see e.g., FIG. 1) are stored in secured transaction database on the central server 104. Any canonical entity 108 that accesses these data objects possesses a non-authoritative copy of the data; they are able to access the data, but cannot modify it themselves. Each data object 136 in the secure transaction database is associated with certain characteristics that are properties of the secured transaction.

A secure transaction is initiated when a player's peer machine 106 takes an action that would alter or transfer a secure data object 136. Any canonical entities 108 involved in the transaction send a message to the central server 104, requesting that the transaction be executed. Once the central server 104 has received all required instructions from the appropriate canonical entities 108 for the transaction (and by extension has received authorization that the transaction is "legal"), it performs the entire transaction as an atomic action. Finally, the central server 104 sends an update message to each canonical entity 108 that holds a copy of the data objects 136 involved, informing them of the changes. [Though it may be tempting to designate every transaction as secure, it is important to realize that the DEX system 100 intrinsically offers substantial protection against cheating and attacks through its basic architecture. The secure transaction system exists largely as a method to provide final assurance against denial of service (DOS) attacks and so-called "duping exploits" for particularly valuable data objects 136. The secure transaction system is designed to govern infrequent, crucial data transfers, such as extremely valuable item trades or avatar "death", not with the everyday tracking of avatar statistics or randomly generated data objects.]

The DEX system 100 is designed to require as little change to the development environment and development process of simulations as possible, while offering substantial cost savings and additional features. As such, it presents an interface to simulation developers, e.g., game developers, that resembles the client-server systems that they are familiar with. The DEX system 100, not the developer, is responsible for breaking the world into regions and distributing them to the canonical entities 108, managing the hierarchy 102,
and handling all assessment, assignment, and allocation of peer resources.

To ensure the security of the DEX system 100, all communications between the peer machines 106 in the game can be routed through encrypted network protocols. The DEX system 100 provides these protocols, as well as the cryptosystems that underlie them.

Additionally, the DEX system 100 provides synchronized parallel random number generators to allow different peer machines 106 to synchronously produce identical results.

Finally, the DEX system 100 provides an automated patching structure that allows integration of patches and new releases into the update schedule. As a player logs on, the DEX system 100 can automatically interfaces with a version tracking system on their peer machine 106, via the billing service 122, and on the machine management service 124 to update the user's peer machine 106 with any new game data that they are required and authorized to have.

Because the canonical entities 108 and the hierarchical entities 110 are composed of the peer machines 106 as peers, the DEX system 100 scales extremely well; when new users log in they bring additional resources with them. Thus, supporting additional users does not require significant additional centralized resources. In the event of unanticipated loads, the DEX system 100 scales immediately, forming new entities and assigning regions of the game world to them "on the fly."

The peer machines 106 perform three crucial roles in the DEX system 100: they support players as users of the system, as members of the canonical entities 108, and as members of the hierarchical entities 110. The player's peer machines 106 may run only the game client, and not be members of any canonical entities 108 or hierarchical entities 110. More typically, however, the player's peer machines 106 will be members of the canonical entities 108, and so these machines will simultaneously run the canonical entity as well as the simulation or game client. The members of the hierarchical entities 110 are also players, so these machines will simultaneously run the hierarchy entity as well as the simulation or game client.

It is safe to assume that there will be significant differences in the computing capabilities of the peer machines 106, some being much faster and more powerful than others. These differences will be taken into account in deciding which roles each peer machine 106 plays, so as not to over-burden any of the users' peer machines 106 or their bandwidth connections and thereby detract from their playing experience.

The load on normal player peer machines 106 is the same as it would be in a client-server game. The bandwidth load for members of a canonical entity 108 can easily be
handled over a modem. Their processor load is up to four times the minimum required to run
the typical game application; fortunately, this is approximately the same as the normal
recommended system level.

The peer machines 106 in the hierarchy 102 have a negligible increase in their CPU
load, and could also be run over a modem, but the DEX system 100 can take advantage of
broadband connections if they are available. Even a small number of broadband users (2-5%
of the audience) would allow the system to free up a tremendous number of slower peer
machines 106 to do nothing other than play the game.

This discussion now turns to the security and robustness of the DEX system 100.

"Spoofing," or impersonating a member of the system, is an area of security that has been a
focus of the computer security community for decades, and the current best practices in the
area are sufficient for the needs of the DEX system 100.

Attacks against the central server 104 is only a minor concern for the DEX system
100. Because the game's publisher owns these computers, they will likely be placed in a
managed hosting facility. Doing so allows the publisher to pay for their expected bandwidth
usage, reduces the danger of DOS attacks, and enables strong measures to protect those
machines against outside intrusion. It is important to note that a DOS attack of the central
server 104 is no easier to perpetrate than such an attack against servers for current
client/server games, and the benefits to a malicious user of accomplishing this attack are
substantially smaller than they are under a current client/server paradigm.

In a targeted dos attack, the attacker wants to attack a specific target, and has the
resources necessary to flood that target's bandwidth and prevent it from being able to
communicate with the rest of the system. Doing so requires that the attacker know the IP
address of his target. These types of attacks can be used to target another player (so that the
attacker could kill his character while he's locked out), to target a server (to prevent that
server from doing anything, intractably slowing down the entire game), or to target any other
computer for which the IP address is known.

In the DEX system 100, a player's peer machine 106 will have access to very few IP
addresses, and will have nothing to gain by targeting those addresses. In particular, it will
only have IP addresses for the canonical entity 108 running his part of the game world, for
players in the game for which that peer machine 106 is a member of a canonical entity 108,
and for other peer machines 106 in the hierarchy 102 that communicate with him directly. If
that player is a member of a canonical entity 108, he will additionally have the addresses for
the other components of his canonical entity 108, and he will have the address of one member
of each entity with which his entity is, or has recently, communicated.

There is very little reason for a player to use a DOS attack against his own canonical entity 108, because the result would only disrupt his own game and that of a few other players in his immediate area. Even if he does target his own canonical entity 108, this will only disrupt the game experience for a small number of players (approximately 10-12), including himself.

There would be very little reason to disrupt the other members of a canonical entity 108 for which he were a member, because doing so would only affect players in a totally different part of the world from his own avatar, making it very unlikely that they would be able to affect anyone that they care about.

Targeting a member of the hierarchy 102 will not accomplish anything, because the game is designed to accommodate those machines suddenly exiting the DEX system 100 anyway. Targeting a single member of another canonical entity 108 will not do sufficient damage to destroy its usefulness.

Each peer machine 106 actively participating in the DEX system 100 participates in several active communications channels over which traffic constantly flows. Each entity in the DEX system 100 votes frequently, say, every 0.2 seconds, and each player's peer machine 106 sends commands up to its canonical entity 108 and receives outputs from it. As such, because the entire DEX system 100 exists in a web of virtual "keep-aliases"; it is very simple to detect increased network latency and unexpected disconnection.

If a peer machine 106 in the DEX system 100 disappears it will be detected via passive means when it fails to respond for some period of time. This period of time can be arbitrarily set to minimize the false positives (times when peer machines 106 are actually still online) while minimizing the effects of the true positives (errors that occur while waiting to replace a peer machine 106 that has gone offline). A node peer machine 106 that goes offline may have been serving as a player, a member of a canonical entity 108, or a member of a hierarchical entity 110.

In the role of player, it is unnecessary to take any action beyond dropping the player's peer machine 106 from the game in which they are playing in a timely fashion.

In the role of canonical entity 108, the remainder of the canonical entity 108 quickly detects that the peer machine 106 has gone offline and informs its parent hierarchical entity 110 of this problem. That hierarchical entity 110 contacts the machine management service 124 in order to find an available new peer machine 106 to step into the canonical vacancy.

If the peer machine 106 that disappeared was serving as a member of a hierarchical
entity 110, its parent in the hierarchy 102 will notice its disappearance. That parent will contact the rest of its own hierarchical entity 110 (or the root central server 104) and request a replacement.

As outlined above, replacement of a peer machine 106 in an entity involves updating address and secure key information for all involved peer machines 106. Thus, the DEX system 100 can continue functioning even in the face of unexpected disconnections.

Before concluding, the following is a brief discussion of background and underlying technology used by the DEX system 100 draws upon several published algorithms and techniques used for cryptology, random number generation, and game synchronization. This section briefly summarizes the relevant technologies.

Modern online games usually make use of publicly available cryptosystems and cryptology packages for communications security. Because these packages are designed for flexibility in multiple applications, they are primarily public-key systems, which have high computational overhead. By designing a networking security layer specifically for the DEX system 100, we are able to make use of substantially faster private-key cryptosystems, such as Rijndael, to allow the system to encrypt its data transmissions in real time. By combining the speed of private-key systems with a session-key paradigm, we ensure that communications in the DEX system 100 will be secure. Additional techniques such as message signing and hashing allow us to prove the identity of a message’s sender and detect errors.

The DEX system 100 relies on three families of cryptology algorithms. It uses key agreement protocols, private and public-key cryptosystem, and message digesting and signing algorithms. Once two nodes (e.g., peer machines 106) have mutually decided upon a key utilizing a key agreement protocol, a secure cryptosystem must be used to protect the communications between the nodes. Two forms of cryptosystem are available for this purpose: public- and private-key systems. Finally, the message digesting and signing algorithms are used in order to ensure the integrity of the communications themselves. These algorithms ensure communications security and data integrity.

Key agreement protocols allow two parties (e.g., peer machines 106) to determine a secret key over a non-secure channel, such as the Internet. Once these two parties have determined the secret key, they may use normal encryption techniques to communicate securely. Even if an attacker is able to intercept and read all messages in the key agreement exchange, they will not be able to discover the secret key agreed upon. Hence, the newly-formed channel will be completely secure.
Currently, the Diffie-Hellman algorithm is one of the better-known algorithms for key agreement. This algorithm consists of two transmissions, one from each end of the new secure channel. By routing these transmissions through a trusted third party, each end of the channel can verify that the transmission came from the appropriate computer.

Key agreement protocols allow private-key cryptosystems to be used as flexibly as public-key systems. Traditionally, public-key systems have been superior in that two parties could communicate their public keys to each other across a non-secure channel, then communicate without fear of interception. Key agreement systems allow private-key cryptosystems to be used in the same way.

Private-key cryptosystems, also known as symmetric cryptosystems, utilize a single key that is known to the sender and the receiver, and must be kept secret from all third parties. These cryptosystems are characterized by their high speed and small size: they are very quick to implement. In the DEX system 100, private-key cryptosystems provide communications security between the peer machines 106 by encrypting the data before it flows over insecure networks.

A prime example of a private-key system, and one which the inventors consider suitable for protecting the communications within any DEX system 100, is Rijndael. Rijndael is the National Institute of Science and Technology's (NIST) proposed replacement for DES (Digital Encryption System). Rijndael was selected for two reasons: it is secure, and it can be used to encrypt data faster than that data can be sent over a LAN, with very low CPU load: Rijndael can be used to encrypt data for transmission in real time. At this time, the inventors consider communications between the peer machines 106 of the DEX system 100 to be adequately protected if Rijndael is used.

Public-key cryptosystems are known as asymmetric cryptosystems because they utilize two keys — one private and one public. In public-key cryptology, only the message recipient knows the private key while the public key may be distributed freely. Messages may be encrypted with the public key, but may only be decrypted with the private key, allowing far greater flexibility than private-key cryptosystems.

The downside of public-key cryptosystems is their low speed and long key lengths.

Public-key cryptosystems are substantially weaker for a given key length than private-key systems. Furthermore, public-key cryptosystems require a substantially longer time to encrypt and/or decrypt a message than their private-key counterparts. For this reason, public-key systems work poorly with long messages.

Because of the disparity in speed between private-key and public-key cryptology,
communications in the DEX system 100 are primarily secured through the faster private-key systems. Public-key cryptosystems may be reserved for initial logon and identity verification.

In order to verify that a message is authentic, it is necessary to have some form of digital signature attached to the message. As discussed above, public-key systems perform poorly on long messages. As a consequence, the current best practice in this area is to create a message digest and sign that digest using the private key from a public-key cryptosystem. This allows anyone who receives the message to decrypt the digest using the sender's public key. Then, they can apply the digest algorithm to the message and verify that the message originated from the correct person and has not been tampered with in transit. Some good algorithms for this application are the MD-2, MD-4 and MD-5 algorithms for generating secure collision-resistant hashes, which were developed at RSA Labs by Ron Rivest and the RSA public-key cryptosystem for signing the digests.

The superior speed of private-key cryptosystems allows a different technique to be utilized if the number of recipients of the signed message is low. By transmitting a message over a private-key channel, the identity of the sender is automatically confirmed. The recipient then automatically knows that the message originated with the sender.

The DEX system 100 makes use of both of these systems for its voting schemes used by the canonical entities 108 and hierarchical entities 110. The simulation data that is to be voted on is transformed into a message digest using a standard collision-free hash. This hash is then sent to other members of the entity over a private-key channel. The private-key channel allows the recipient of the message to confirm its sender, and the collision-free hash allows them to determine whether or not the sender had the same world state as the recipient.

One of the fundamental characteristics of the DEX system 100 is its use of "mirrored execution," or multiple peer machines 106 performing exactly the same calculations in parallel. In this situation, each peer machine 106 performing the calculation must generate precisely the same sequence of random numbers, with no deviations. This task is even further complicated when the peer machines 106 are simultaneously executing several independent game processes. So, in order to keep the game execution synchronized, it is important to use a random number generator that works well in a multiprocessing environment. Additionally, this generator must be fast and difficult to predict, even if a hacker knows the sequence of its past outputs.

Fortunately, advanced random number generators have been developed which solve both of these problems. These random number generators are available in many designs: some are sufficiently unpredictable to be classified "crypto-secure," while others are faster
and more random than the hardware random number generators built into all desktop computers. A random number generator appropriate for the application may be inserted as a modular component into the DEX system 100, providing parallel identical random number generation at a useful speed. [All computer random number generators are actually pseudorandom number generators. Pseudorandom numbers are like random numbers in that they hold the same properties regarding distribution and frequency, but are wholly deterministic -- knowing all the numbers that have gone before allows the prediction of the next number. While pseudorandom numbers are sufficiently random to replace truly random numbers in computational problems, the DEX system 100 can also take advantage of their deterministic nature.]

One such random number generator was designed by Daniel Pryor of Los Alamos National Laboratory and can be used to generate several independent series of random numbers in a multiprocessing environment. Thus, interactions between different parts of the simulation can be separated from each other, and differences between client computers can be compensated for. The use of Pryor (or a similarly powerful random number generator) allows complex executions to be synchronized successfully. Unmodified, Pryor's generator is sufficiently fast for the purposes of the DEX system 100, but it is too predictable. By modifying the algorithm slightly, so that some of its outputs are sent to the peer machine 106 and some only modify the internal state of the generator, it is made difficult to predict.

The ability to synchronize execution on multiple peer machines 106 is important to the functioning of the DEX system 100. Suitable methods used for maintaining synchronization may be drawn from two existing techniques that have been used in games in the past.

Microsoft's ENSEMBLE STUDIOS achieves impressive synchronization in its AGE OF EMPIRES series of games using its NET.LIB technology. This system allows multiple peers to execute the same code at precisely the same time while taking into account actions by all players involved in the game (currently up to eight), thus, enabling fully synchronized execution -- and, therefore, meaningful comparisons between peers.

A different technique is exemplified by EPIC MEGAGAMES in the UNREAL game engine. This provides a method for synchronization between client and server machines while ensuring very quick responses for the players. This technique permits excellent gameplay, but does not allow for multiple machines to synchronize with each other independent of servers.

These technologies have some problems that influence their use in the DEX system 100. In the ENSEMBLE engine, a first problem is that its random number generator does not
work well in a multiprocessing environment. Secondly, its cryptosystem is not sufficiently fast to allow encryption of all communications. Likewise, EPIC's approach does not deal well with peer systems in which there is no authoritative server. Therefore, as described above, the DEX system 100 draws upon the techniques pioneered in these earlier games and improves upon them by integrating them with modern random number generators and cryptosystems.

While various embodiments have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of the invention should not be limited by any of the above described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

INDUSTRIAL APPLICABILITY

The present distributed execution system, DEX system 100 is well suited for application in simulations, generally as well as in specific variants such as scientific modeling, multiplayer games, and analyzing large data sets. Current client-server architectures suffer from high cost and management burden and current peer-to-peer applications suffer from poor security and scalability. The DEX system 100 offers the strengths of both client-server and peer-to-peer architectures, yet overcomes these drawbacks.

As has been discussed herein, the DEX system 100 is particularly able to execute multi participant simulations where preserving data security and integrity are important, even in the face of participant efforts and collusion to undermine the data and the simulation itself.

The DEX system 100 is a hybrid system. It functions somewhat like a traditional client-server system, except that instead of handling execution on centralized servers it distributes execution over many peer servers, i.e., participant's peer machines 106. In this manner, the peer machines 106 execute tasks for other peer machines 106 and scalability is simplified. As participants join the simulation they inherently bring additional execution capacity to it.

The DEX system 100 also provides levels of performance, reliability, data integrity, and security equivalent to existing client-server systems. Yet it significantly reduces the cost and risk of developing simulations, and it further significantly reduces the costs of operating and supporting such simulations.

For all of its benefits, however, the DEX system 100 is relatively easily implemented with conventional skills and technologies, once the teachings herein are grasped. The
invention employs several prior art elements, including the present inventor's own prior invention of PDP 10 and published algorithms in the categories of cryptology, random number generation, and secret sharing.

For the above, and other, reasons, it is expected that the DEX system 100 of the present invention will have widespread industrial applicability. Therefore, it is expected that the commercial utility of the present invention will be extensive and long lasting.
What is claimed is:

1. A system for distributing execution of a simulation, the system comprising:
   a central server;
   a plurality of peer machines each able to act as a participant in and an executor of
   regions of the simulation;
   one or more hierarchical entities including multiple said peer machines;
   one or more canonical entities including multiple said peer machines;
   said central server, said hierarchical entities, and said canonical entities being
   organized to form a hierarchy having a tree structure wherein said central
   server is a root, said hierarchical entities are branches, said canonical entities
   are leaves, said root is defined as upward, and said leaves are defined as
   downward;
   within each said canonical entity, each said peer machine executes the same said
   region of the simulation, intercommunicates its result within its said canonical
   entity, performs comparison of all said results of its said canonical entity,
   derives a canonical vote based on its said comparison, and communicates said
   canonical vote up said hierarchy;
   within each said hierarchical entity, each said peer machine derives a present
   hierarchical vote based on said canonical votes and any other said hierarchical
   votes received from down said hierarchy, and communicates said present
   hierarchical vote up said hierarchy; and
   said central server assigns said regions to said hierarchical entities and said canonical
   entities, said central server assigns said peer machines to be members of said
   canonical entities and said hierarchical entities such that no said peer machine
   is both a said participant and a said executor with respect to any one of said
   regions, said central server directs said hierarchical entities and said canonical
   entities to execute said regions, and said central server integrates said regions
   into the simulation, thereby distributing execution of the simulation among
   instances of said peer machines acting as said executors to benefit instances of
   said peer machines acting as said participants.

2. The system of claim 1, wherein the simulation is a game and said participants are
   players of said game.
3. The system of claim 2, wherein said game is a massively multiplayer online game (MMOG).

4. The system of claim 1, wherein said central server includes a plurality of server computers.

5. The system of claim 1, wherein said central server operates at least one member of the set consisting of authentication services, billing services, machine management services, tree management services, patch management services, persistent data services, and event management services.

6. The system of claim 1, wherein said hierarchy intercommunicates via the Internet.

7. The system of claim 1, wherein said peer machines include personal computers.

8. The system of claim 1, wherein said at least one of either said hierarchical entities or said canonical entities includes an odd number of said peer machines, thereby facilitating tie vote avoidance.

9. The system of claim 1, further comprising:
   a spot-check entity including multiple said peer machines, wherein said spot-check entity is paired in said hierarchy with and performs the same tasks as a suspect said canonical entity;
   said canonical votes derived by said spot-check entity are spot-check votes; and
   said hierarchical entities prioritize said spot-check votes over said canonical votes derived by said suspect canonical entity.

10. The system of claim 1, further comprising:
    a spot-check entity including multiple said peer machines, wherein said spot-check entity is paired in said hierarchy with and performs the same tasks as a suspect said hierarchical entity;
    said hierarchical votes derived by said spot-check entity are spot-check votes; and
said hierarchical entities prioritize said spot-check votes over said hierarchical votes derived by said suspect hierarchical entity.

11. A method for distributing execution of a simulation, the method comprising the steps of:
(a) constructing a hierarchy having a tree structure from a central server, one or more hierarchical entities including multiple peer machines, and one or more canonical entities including multiple peer machines, wherein said central server is a root, said hierarchical entities are branches, said canonical entities are leaves, said root is defined as upward, and said leaves are defined as downward;
(b) segmenting the simulation into regions suitable for said peer machines to act as a participant in or an executor of;
(c) assigning said regions to said hierarchical entities and said canonical entities;
(d) assigning said peer machines to be members of said canonical entities and said hierarchical entities, wherein no said peer machine is both a said participant and a said executor with respect to any one of said regions;
(e) directing said peer machines within respective said canonical entities to execute the same said region of the simulation, to intercommunicate its result within its said canonical entity, to perform comparison of all said results of its said canonical entity, to derive a canonical vote based on its said comparison, and to communicate said canonical vote up said hierarchy;
(f) directing said peer machines within respective said hierarchical entities to derive a present hierarchical vote based on said canonical votes and any other said hierarchical votes received from down said hierarchy, and to communicate said present hierarchical vote up said hierarchy; and
(g) integrating said regions into the simulation, thereby distributing execution of the simulation among instances of said peer machines acting as said executors to benefit instances of said peer machines acting as said participants.

12. The method of claim 11, wherein said simulation is a game and said participants are players of said game.
13. The method of claim 12, wherein said game is a massively multiplayer online game (MMOG).

14. The method of claim 11, further comprising:
   (h) operating centralized services within said central server to facilitate providing the simulation to said participants, wherein said centralized services include at least one member of the set consisting of authentication services, billing services, machine management services, tree management services, patch management services, persistent data services, and event management services.

15. The method of claim 11, further comprising:
   (h) intercommunicating within said hierarchy via the Internet.

16. The method of claim 11, further comprising:
   (h) including in at least one of either said hierarchical entities or said canonical entities an odd number of said peer machines, thereby facilitating tie vote avoidance.

17. The method of claim 11, further comprising:
   (h) assigning said peer machines to be members of a spot-check entity; (i) pairing said spot-check entity in said hierarchy with a suspect said canonical entity such that said spot-check entity performs the same tasks as said suspect canonical entity and said canonical votes derived by said spot-check entity are spot-check votes; and (j) prioritizing said spot-check votes over said canonical votes derived by said suspect canonical entity.

18. The method of claim 11, further comprising:
   (h) defining one said canonical entity or one said hierarchical entity to be a suspect entity and any said canonical votes or said hierarchical votes produced thereby are suspect votes; (i) assigning some said peer machines to be members of a spot-check entity; (j) pairing said spot-check entity with said suspect entity in said hierarchy;
(k) performing the same tasks as said suspect entity, wherein any said canonical votes or said hierarchical votes produced thereby are spot-check votes;

(l) or said suspect hierarchical entity and said canonical votes or said hierarchical votes derived by said spot-check entity are spot-check votes; and

(m) prioritizing said spot-check votes over said suspect votes.
FIG. 1 (background art)
FIG. 3
System Overview
INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 9/44, 17/00; H01J 13/00
US CL : 709/205,204; 717/170
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/205,204; 717/170

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>US 5,812,773 A (NORIN) 22 September 1998, see whole document.</td>
<td>1-18</td>
</tr>
<tr>
<td>Y</td>
<td>US 5,948,057 A (BERGER et al.) 07 September 1999, see whole document.</td>
<td>1-18</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of Box C.

* Special categories of cited documents:
  
  "A" document defining the general state of the art which is not considered to be of particular relevance
  
  "E" earlier application or patent published on or after the international filing date
  
  "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  
  "O" document referring to an oral disclosure, use, exhibition or other means
  
  "P" document published prior to the international filing date but later than the priority date claimed

  "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

  "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

  "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

  "&" document member of the same patent family

Date of the actual completion of the international search
04 April 2003 (04.04.2003)

Date of mailing of the international search report
25 APR 2003

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
Ayaz R. Sheikh

Telephone No. 703-305-9648

Form PCT/ISA/210 (second sheet) (July 1998)
Continuation of B. FIELDS SEARCHED Item 3:
East search terms:
central server, peer machine, hierarchical tree, hierarchical entities, canonical entities, simulation, distribution system.