RZ 3500 (# 99436) 06/30/03
Computer Science 14 pages

Research Report

Communication Architectures for Massive Multi-Player Games

Daniel Bauer, Ilias Iliadis, Sean Rooney, and Paolo Scotton

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Communication Architectures for Massive Multi-Player Games

Daniel Bauer, Ilias Iliadis, Sean Rooney, Paolo Scotton
IBM Research, Zurich Laboratory Säumerstrasse 4
8803 Rüschlikon, Switzerland

Abstract

Massive multi-player games are characterized by a large number of participating players. It is therefore essential that an appropriate communication architecture is deployed in order to support an ever growing number of players. Several such architectures have been proposed, including client-server and peer-to-peer architectures. In this paper, we propose a systematic method to assess the scalability of different architectures in order to identify the most appropriate one for specific game types. The model proposed is very general in that it covers centralized, distributed, and hybrid architectures and it is applied to the client-server, peer-to-peer and the newly introduced federated peer-to-peer architecture. Quantitative expressions that capture the effect of various game types are derived, and the trade-offs among the architectures are identified.

I. Introduction

The term Massive Multi-Player On-Line Game (MMPOG) has come to denote games with a large number of participants played over the Internet, for example the EverQuest role-playing game [13]. A key challenge in building such games is the design of a communication architecture that is able to support thousands of players. A large number of different architectures have been proposed, which can be classified into *client-server* and *peer-to-peer* architectures. The client-server architecture is centralized, whereas the peer-to-peer architecture is fully distributed. Later in this paper, we present our own architecture, which is a hybrid of the client-server and peer-to-peer architectures and which we call *federated peer-to-peer* architecture.

Client-server architectures are favored by game middleware providers, as they allow a very tight control of the game state. In these architectures, the central server executes the entire game logic, whereas the clients are terminals that visualize the game state as provided by the server. Examples of such architectures are described in [8], [10]. As most of these architectures are proprietary, there are no figures available with respect to the scalability of these systems. An investigation of network traffic in a client-server architecture has been carried out in [2].

Peer-to-peer architectures are used for real-time, interactive military simulations and war-gaming. The DIS standard [1] is a set of protocols designed for a peer-to-peer architecture. The simulation architecture consists of several simulators that exchange information using a broadcast mechanism. Simulators may simulate several objects and/or participants. The original DIS approach is not scalable to a large number of simulators; it was primarily used for small-scale, local simulations in LANs. MiMaze [6] is a peer-to-peer multi-player game developed for researching the peer-to-peer architecture. It follows the DIS architectural concept without using specifically the DIS protocol. Scalability of MiMaze is limited; the game supports fewer than 100 players.

A more scalable variant of the peer-to-peer architecture is used in the NPSNET 3D vehicle simulator [9]. NPSNET is based on the DIS protocols, but uses a network architecture that partitions the virtual environment into spatial, temporal, and functionally related interaction groups. The spatial partition divides the virtual space into a set of hexagons. Simulated objects subscribe to interaction groups representing

the hexagons close to their virtual location. As objects move, the set of interaction groups to which they are subscribed changes dynamically. Temporal interaction groups are used to cover larger virtual areas up to the entire simulation space. However, information is sent at a much lower frequency than in the spatial interaction groups. Functional interaction groups are used to enable the communication among objects of the same type, independently of their virtual location. The partitioning of the virtual environment into interaction groups and the orchestration of these groups are done by an Area Of Interest Manager (AOIM). The AOIM is part of every simulator and runs itself in a distributed fashion. The core simulators are not necessarily aware of the partitioning, as the DIS protocols runs unchanged. Although the NPSNET architecture has introduced the notion of interaction groups to support large-scale simulation, no quantitative assessment on the scalability has been reported.

In the efforts mentioned above, no means are provided to assess scalability issues. In the cases where scalability figures are given, these figures are tied to a specific scenario and game. In this paper, we present a quantitative model for the evaluation of the scalability of different game architectures. We define cost functions to quantify the amount of processing and networking resources a given architecture requires. The effect of various game types can be captured by setting the parameters associated with the cost functions to appropriate values. We apply these cost functions to the client-server, peer-to-peer and federated peer-to-peer architectures, and discuss the trade-offs among them for a range of parameter values.

This paper is organized as follows. First, related work is discussed in Section II. A generic game model that captures the costs of various game operations required is presented in Section III. In Section IV-A, this model is applied to the client-server, peer-to-peer and federated peer-to-peer architectures in order to derive the architecture-specific cost functions. Then, in Section V, we evaluate and compare the architectures considered based on the cost functions obtained. Concluding remarks follow in Section VI.

II. Related Work

Baughman et al. [4] explore the vulnerability to cheating in client-server and peer-to-peer architectures. They introduce a game model in which the entire game state is composed of entity states that are administered by the various components of an architecture. This model is then used to describe several cheats and their impact on the different architectures. The paper proposes a new protocol that counters several types of cheats. Our model is based on the model proposed by Baughman et al. in [4], but has been extended in two ways. First, we introduce resource cost functions that allow us to assess resource usage in various scenarios. Second, we model game state partitioning by introducing the concept of interaction groups.

A review of the techniques used to build MMPGs and virtual environments is given in [12]. This review introduces a model that describes how messages are relayed between local and remote nodes. The model is used to discuss aspects of game consistency and responsiveness quantitatively.

Interest management based on IP multicast groups and end-system filtering were compared in [7]. The paper shows that the current Internet architecture based on IP multicasting is not suitable for large-scale applications because it can't manage multiple multicast groups efficiently. The paper also argues that interest management based on a multicast architecture is preferable to interest management based on end-system filtering. The results have been derived using a model for game network traffic to assess different scenarios. While the model of Levine et al. [7] primarily focuses on the effects of interest management on networking resources, the model we proposed captures networking and processing resources of entire game architectures.

The effect of different grouping strategies on the overall cost of state dissemination was investigated in [14]. These two grouping strategies are discussed: region-based and entity-based grouping. Our model does not consider different grouping strategies; we assume that a grouping strategy is used in which each group has the same average size. The details of how groups are formed are not considered in our model.

III. GAME MODEL

We characterize games using a model initially proposed by Baughman et al. in [4] and extend this model with resource-cost functions and the concept of interaction groups. The game is a large state machine that consists of entity states. Each entity is composed of several in-game objects that are controlled by a player. Entity states are periodically updated. During each game period, each player issues exactly one command that is taken into account to update the corresponding entity state. Each player controls his or her objects, but these objects also interact with each other. An entity state therefore is updated based on the input from the player as well as the state of other entities. Entities that interact with each other form a group. An interaction group typically consists only of a small subset of all entities in the game, as interaction only happens locally. Interaction groups may overlap in the sense that a single entity might be part of several interaction groups.

We now introduce the following parameters:

- n - the number of entity states in the game.
- N - the number of interaction groups in the game ($1 \leq N \leq n$).
- h - the average number of interaction groups that a player joins ($h \geq 1$).
- g - the average size of an interaction group in the game. It holds that $g = hn/N$, which implies that $g \geq n/N$.

We model two types of resources that are used by the game, namely, internal processing capabilities and network processing capabilities used for transmission and reception. Internal processing capabilities are used to process input from players, to correlate entity states, and to visualize the state of an entity to the player. We assume that the visualization is executed on each player's computer with constant cost, which is therefore no longer considered in the remainder of the paper. Transmission and reception processing capabilities are used to transport user-input events and state-update messages. It is assumed that the network cost corresponding to the bandwidth and capacity usage is proportional to the network processing cost, and it is therefore not considered separately.

Let us now introduce the cost of operations on the resources as follows:

- $p_i(x)$ - The cost of processing x user-input events.
- $p_c(x)$ - The cost of correlating x entity states of an interaction group.
- $t_i(x)$ - The processing cost of transmitting a user-input event to x destinations.
- $r_i(x)$ - The processing cost of receiving x user-input events.
- $t_u(x)$ - The processing cost of transmitting an entity-state update to x destinations.
- $r_u(x)$ - The processing cost of receiving x entity-state updates.

We proceed by considering the following relations:

- $p_i(x) = f_i^{(i)} x$, i.e. a linear cost function.
- $p_c(x) = [x(x-1)/2] f_c^{(i)}$, where the term in brackets is the number of all entity pairs to be correlated within an interaction group of size x.
- $t_i(x) = m(x) f_i^{(n)}$, where $m(x)$ is equal to one in the case of multicast and equal to x in the case of unicast.
- $r_i(x) = x f_i^{(n)}$, i.e. a linear cost function.
- $t_u(x) = m(x) f_u^{(n)}$.
- $r_u(x) = x f_u^{(n)}$, i.e. a linear cost function.

The above costs are expressed in processing units with the factors of the form $f^{(i)}$ and $f^{(n)}$ referring to the internal and network processing operations, respectively. Note also that the factors $f_i^{(i)}$, $f_c^{(i)}$, $f_i^{(n)}$, and $f_u^{(n)}$ are only game dependent. In this paper we consider the case of unicast, such that $m(x) = x$.

In the next section we will consider the following game architectures: client-server, peer-to-peer, and federated peer-to-peer architectures. For the client, server, and peer entities, the average cost for internal

processing C_i is of the form,

$$C_i = p_i(x_A) + B_A\, p_c(g_A)\ ,$$

where the number x_A of the user-input events, the constant B_A, and the average size g_A of the interaction groups depend on the architecture chosen. Similarly, the average cost for networking C_n is given by

$$C_n = t_i(x_{A,1}) + r_i(x_{A,2}) + t_u(x_{A,3}) + r_u(x_{A,4})\ ,$$

where the variables $x_{A,j}$ ($j = 1, 2, 3, 4$) depend on the game architecture chosen.

IV. Basic Game Architectures

Here we consider the model established in Section III and apply it to the client-server and the peer-to-peer architectures in order to obtain the resource (average) cost functions. We also apply it to the new architecture proposed in [11], called the *federated peer-to-peer* architecture, which combines elements from both the client-server and the peer-to-peer architectures.

A. Client-Server Architecture

In the client-server architecture, the entire game state is kept on a central server. There are two variations of the architecture. The approach using central input processing is shown in Figure 1. Processing of the game state is done entirely on a central server. Client machines are terminals that display the game state and send the player's input events to the central server.

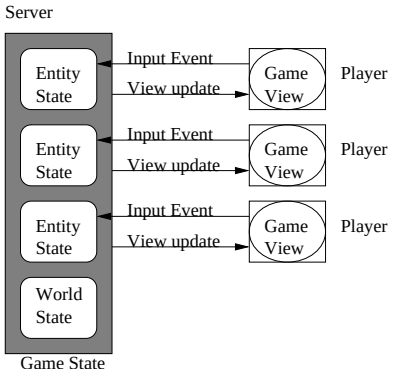

Fig. 1. Client-Server Architecture with Central Input Processing

During each period in the game, the server uses resources for receiving, processing and transmitting input events and state updates, as shown in the table below. Each of the n clients sends its input events to the server, where they are used to compute entity-state updates. The server then correlates the entity states in all N interaction groups. The resulting state updates are subsequently returned to the clients. This leads to the following cost functions:

Server Costs for Central Input Processing	
C_n	$r_i(n) + t_u(n) \quad = n(f_i^{(n)} + f_u^{(n)})$
C_i	$p_i(n) + Np_c(g) \quad = f_i^{(i)} n + N f_c^{(i)} g(g-1)/2$
Total	$n(f_i^{(n)} + f_u^{(n)} + f_i^{(i)}) + N f_c^{(i)} g(g-1)/2$

The clients require resources for transmitting input events and receiving state updates[1]. The resulting costs are shown in the table below:

[1]Visualization and rendering are not considered, cf. Section III.

Client Costs for Central Input Processing	
C_n	$t_i(1) + r_u(1) \quad = f_i^{(n)} + f_u^{(n)}$
C_i	0
Total	$f_i^{(n)} + f_u^{(n)}$

The second approach is called client-server architecture with local input processing, shown in Figure 2. Each client is responsible for maintaining its entity state. The player's input is processed locally, to the greatest extent possible, and a state update is subsequently sent to the central server. The central server coordinates the state updates received from all the players, resolves conflicts, and redistributes the definite entity states. In contrast to the central input processing approach, the processing of the player's input is done on client machines.

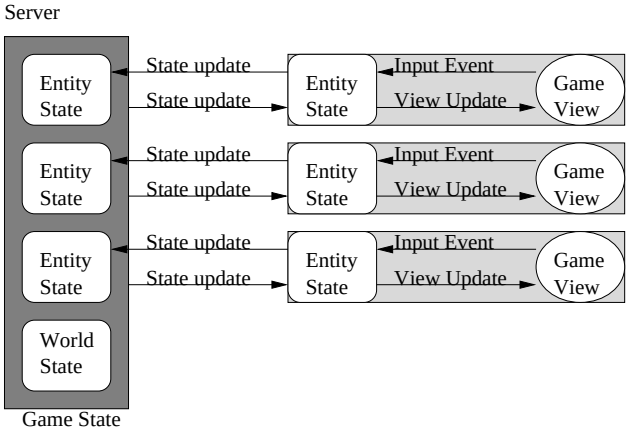

Fig. 2. Client-Server Architecture with Local Input Processing

In the local input processing architecture, the server receives state updates from all entity states as the clients process the corresponding user inputs. The server then correlates the entity states within each interaction group and returns the resulting entity states to the clients. The corresponding costs are shown below.

Server Costs for Local Input Processing	
C_n	$t_u(n) + r_u(n) \quad = 2nf_u^{(n)}$
C_i	$Np_c(g) \quad = Nf_c^{(i)}g(g-1)/2$
Total	$2nf_u^{(n)} + Nf_c^{(i)}g(g-1)/2$

Client Costs for Local Input Processing	
C_n	$t_u(1) + r_u(1) \quad = 2f_u^{(n)}$
C_i	$p_i(1) \quad = f_i^{(i)}$
Total	$2f_u^{(n)} + f_i^{(i)}$

For the client-server architecture, we assume that maintaining the interaction groups incurs negligible cost. As the maintenance is done on the server only, no communication costs arise. Furthermore, we assume that maintaining the interaction groups is essentially a free by-product of correlating the entity states.

B. Peer-to-peer Architecture

In the peer-to-peer approach, no central server exists. The game state is distributed among several peers, as shown in Figure 3. A peer maintains the entity states of all interaction groups to which the player is

subscribed. In the extreme case of only a single interaction group, the entire game state is replicated on all peers. Replicated peer-to-peer architectures are often used in games with a small number of players, such as in the Age of Empires series [5].

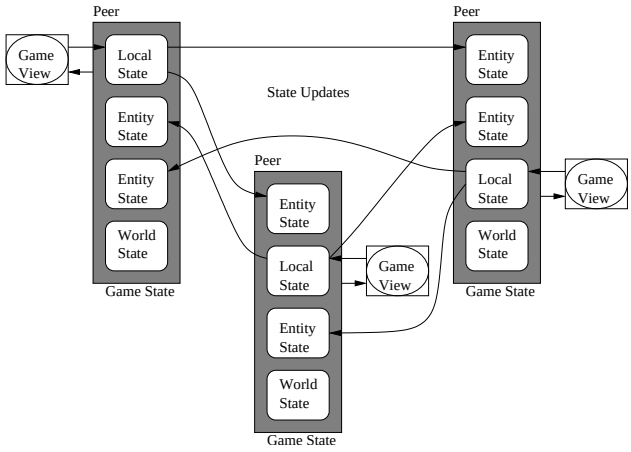

Fig. 3. Peer-to-Peer Architecture

Two versions of the architecture exist, analogous to the client-server architecture. In local input processing, peers use the input event to update the local entity state and transmit a state update to all other peers. In the case of distributed input processing, peers transmit input events to all other peers, which then compute the entity-state updates.

The resource costs are composed of the cost for keeping the entity states up-to-date as well as the cost for maintaining the interaction groups. The latter is the price to be paid for adopting a distributed solution. We first start by computing the game-state processing cost for a single peer. In the local input processing version, this cost consists of processing the user input, sending the local state update to $g-1$ peers, receiving state updates from the other peers, and correlating all g state updates. This occurs per interaction group, and each peer is member of an average number of h interaction groups. Thus, the local state update is sent to $h(g-1)$ peers. Note that in this scenario a neighboring peer may receive multiple updates if it is a member of more than one of the h interaction groups. The resulting costs are shown below:

Peer Costs for Local Input Processing	
C_n	$(t_u(g-1)+r_u(g-1))h \quad = 2hf_u^{(n)}(g-1)$
C_i	$p_i(1)+p_c(g)h \quad = f_i^{(i)}+f_c^{(i)}(g-1)gh/2$
Total	$(g-1)h(2f_u^{(n)}+f_c^{(i)}g/2)+f_i^{(i)}$

In distributed input processing, each peer sends its input event to $g-1$ peers and receives $g-1$ state updates per interaction group. Each peer processes g input events and state updates, including its own event and state update. Also, each peer is a member of an average number of h interaction groups, resulting in the costs shown below:

Peer Costs for Distributed Input Processing	
C_n	$(t_i(g-1)+r_u(g-1))h \quad = (f_i^{(n)}+f_u^{(n)})(g-1)h$
C_i	$(p_i(g)+p_c(g))h \quad = h(f_i^{(i)}g+f_c^{(i)}g(g-1)/2$
Total	$(g-1)h(f_u^{(n)}+f_i^{(n)}+f_i^{(i)}g/(g-1)+f_c^{(i)}g/2)$

The following aspects are important for maintaining the interaction groups.

- A peer discovers that it needs to subscribe to a new interaction group. As in the case of the client-server

architecture, we assume that this is a by-product of correlating the entity states and that there is no cost involved.

- Each peer maintains only the entity states of the interaction groups of which it is a member. When a peer subscribes to a new interaction group, then it needs to receive the entity states of the new interaction group. We assume that each interaction group elects a group leader that provides these states either directly or indirectly. The method of how a group leader is elected is not further considered here, and we assume that the election cost is negligible. Furthermore, we assume that the costs for transferring and receiving a number of x entity states are $r_u(x)$ and $t_u(x)$, respectively.
- As a consequence of the preceding point, each peer needs to know the group leader of each interaction group, that is the group leader advertises its address to all other peers. We assume that the cost for transferring and receiving group leader information is $r_i(1)$ and $t_i(1)$, respectively.
- The cost for leaving an interaction group is negligible. Again, we assume that the decision to leave an interaction group comes as a by-product of the correlation computation.

To compute the cost for maintaining the interaction groups, we introduce a new parameter that captures the dynamic behavior of games. Let d denote the rate at which a peer switches an interaction group during a game period. During each game period, an average number of dn peers switch their interaction group. This in turn implies that for each interaction group an expected number of dn/N peers leave and dn/N peers join the group.

The maintenance cost for an individual peer accounts for the reception and processing of the entity states of the new interaction group, which is given by $r_u(g)$ and $p_c(g)$, respectively. The rate at which a peer switches groups is d, and a peer is a member of an average number of h interaction groups. Therefore the respective costs per game period for joining are $dr_u(g)h$ and $dp_c(g)h$, respectively.

Next we consider the cost for joining an interaction group. On average, there are dn/N peers that join an interaction group. When a new peer joins an interaction group, the group leader transfers the group's g entity states to this new peer, which costs $t_u(g)$. Furthermore, the group leader is a member of an average number of $h = gN/n$ interaction groups. As there are, on average, g peers per interaction group, the cost for joining, assigned to each peer, is given by

$$\frac{dn}{N}\frac{gN}{n}\frac{t_u(g)}{g} = dt_u(g)$$

Now, we consider the cost for distributing the information regarding the group leaders to the peers. We assume that this information is only distributed when the group leader of an interaction group changes. The group leader transmits this information at a rate of d per interaction group to all peers at cost of $t_i(n)$. As a group leader participates in $h = gN/n$ interaction groups and as an interaction group contains g peers, the corresponding cost per peer is

$$\frac{gN}{n}\frac{dt_i(n)}{g} = dNt_i(1)\ .$$

The total maintenance costs for a peer per game period are obtained by summing up the cost terms mentioned in the last three paragraphs:

Peer Interaction Group Maintenance Costs	
C_n	$d(t_u(g) + Nt_i(1) + r_u(g)h) \quad = d(gf_u^{(n)} + Nf_i^{(n)} + gf_u^{(n)}h)$
C_i	$dp_c(g)h \quad = df_c^{(i)}g(g-1)h/2$
Total	$d(gf_u^{(n)} + Nf_i^{(n)} + gh(f_u^{(n)} + (g-1)f_c^{(i)}/2))$

C. Federated Peer-to-peer Architecture

The federated peer-to-peer approach, shown in Figure 4, is a hybrid of the peer-to-peer and the central-server architectures. It promises a lower resource cost per peer than the pure peer-to-peer architecture, and it

does not concentrate all resources on a single server as the client-server architecture does. The key idea is to separate the management of the interaction groups from the normal game operations. Game operations are handled in a peer-to-peer fashion, where the members of an interaction group send entity-state updates to all other peers in the same peer group. To reduce the communication cost incurred by the peers, fast multicast reflectors handle the communication needs of interaction groups. A multicast reflector maintains a list of peer addresses for each interaction group. Peers send state updates to the multicast reflector which in turn distributes these updates to all peers in the interaction group. More aspects of this architecture can be found in [11], and a report on the performance of a multicast reflector implementation is given in [3].

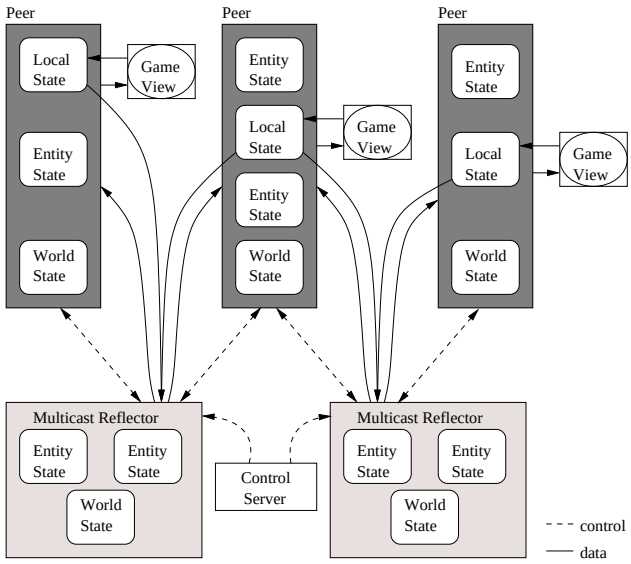

Fig. 4. Federated Peer-to-Peer Architecture

The management of interaction groups is done on a central control server with the support of the multicast reflectors. The control server assigns multicast reflectors to interaction groups; a single multicast reflector might handle several interaction groups. The details of this assignment is game dependent, with one possible and well-known method being to divide the virtual game space into multiple regions. Each region is an interaction group that is assigned to a multicast reflector, described by a mapping of coordinates in the virtual space to a multicast-reflector address. Independently of how the control server does the assignment, the result is a mapping table that maps interaction groups to multicast reflectors. The control server distributes this mapping table to all multicast reflectors, which in turn distribute it to all peers. Peers are able to identify the interaction groups to which they belong based on their local entity state. Using the mapping table, they identify the corresponding multicast reflectors.

Multicast reflectors implement a control protocol that allows peers to easily switch interaction groups. For this purpose, the multicast reflector stores copies of all entity states of an interaction group. Keeping the entity states up-to-date is easy because all entity-state updates are being forwarded by the multicast reflector. If a peer joins a new interaction group, the corresponding multicast reflector adds the peer's address to the address list of the interaction group and transfers the entity states to the newcomer.

The resource cost incurred by a peer during a game period consists of processing the user-input event, sending a single state update to the multicast reflectors of the corresponding interest groups, receiving state updates from all peers, including its own state update, and correlating those. Because a peer is member of h interaction groups, the following costs result:

Peer Costs without Maintenance Costs	
C_n	$(t_u(1) + r_u(g))h \quad = f_u^{(n)}(g+1)h$
C_i	$p_i(1) + p_c(g)h \quad = f_i^{(i)} + f_c^{(i)}(g-1)gh/2$
Total	$h(f_u^{(n)}(g+1) + f_c^{(i)}(g-1)g/2) + f_i^{(i)}$

The resource costs for all multicast reflectors combined are:

Multicast Reflector Costs	
C_n	$N(r_u(g) + gt_u(g)) \quad = Ngf_u^{(n)}(1+g)$
C_i	0
Total	$Ngf_u^{(n)}(1+g)$

The cost for maintaining group memberships of a peer takes into account the following actions:

- A peer detects that it needs to join a new interaction group. We assume that this decision is a by-product of the state correlation and comes for free.
- The peer sends a join message to the corresponding multicast reflector in order to register its address. We assume that this costs $t_i(1)$ for the peer and $r_i(1)$ for the multicast reflector. The multicast reflector subsequently returns the entity states of the interaction group, which costs $t_u(x)$ for the peer and $r_u(x)$ for the multicast reflector. The peer, upon receiving the state updates, needs to correlate them.

Using the switching rate d and the average number of memberships h, the maintenance cost of a peer are:

Peer Maintenance Costs	
C_n	$d(r_i(1) + t_u(g))h \quad = d(f_i^{(n)} + gf_u^{(n)})h$
C_i	$dp_c(g)h \quad = df_c^{(i)}(g-1)gh/2$
Total	$dh(f_i^{(n)} + gf_u^{(n)} + f_c^{(i)}g(g-1)/2)$

Thus, the maintenance cost for all N multicast reflectors combined consists of processing $dNt_i(1)$ join messages and transmitting $dNr_u(g)$ update events.

Multicast Reflector Maintenance Costs	
C_n	$dN(t_i(1) + r_u(g)) \quad = dN(f_i^{(n)} + gf_u^{(n)})$
C_i	0
Total	$dN(f_i^{(n)} + gf_u^{(n)})$

V. EVALUATION

We compare the different game architectures using three different evaluation criteria. First, we consider the scalability in the number of players, and compare the different aspects of the architectures considered. The second part investigates the effect of partitioning on the resource costs. The last part investigates the difference of local input processing versus central/distributed input processing.

A. Scalability and Overall Cost

In order to assess the scalability of the different architectures, we compare the cost functions of the different architectures for growing n, with the other parameters being constant, i.e. the parameters $N, h, f_i^{(n)}$, $f_u^{(n)}, f_c^{(i)}$, $f_i^{(i)}, d$ are considered to be fixed. As N and h are constant, and $g = hn/N$, it holds that g increases proportionally to n. In a first evaluation, the overall cost of the architectures is compared.

1) Client-Server Architecture: The overall cost in the client-server architecture includes the cost of the server plus the cost of all n clients. The total costs of the central input processing architecture is

$$n(f_i^{(n)} + f_u^{(n)} + f_i^{(i)}) + Nf_c^{(i)}\frac{g(g-1)}{2} + n(f_u^{(n)} + f_i^{(n)}), \tag{1}$$

and for the local input processing:

$$2nf_u^{(n)} + Nf_c^{(i)}\frac{g(g-1)}{2} + n(2f_u^{(n)} + f_i^{(i)}). \tag{2}$$

Note that for both local and central input processing the dominating factor of the total server cost is $Nf_c^{(i)}g(g-1)/2$, which implies that the cost growth is on the order of $O(n^2)$.

2) Peer-to-Peer Architecture: In the peer-to-peer architecture, the cost per client of the local input processing variant is:

$$(g-1)h(2f_u^{(n)} + \frac{f_c^{(i)}g}{2}) + f_i^{(i)} \tag{3}$$

and for the distributed input processing:

$$(g-1)h(f_u^{(n)} + f_i^{(n)} + f_i^{(i)}\frac{g}{g-1} + \frac{f_c^{(i)}g}{2}). \tag{4}$$

For large n, $g/g-1$ approaches unity, and Equations (3) and (4) differ by constant terms only. We obtain the processing costs for all peers by multiplying (3) and (4) with n, both equations then take the form

$$n(k + k'(g-1) + k''g(g-1)) \ , \tag{5}$$

where k, k', k'' are constants. As g grows proportionally to n, Equation (5) grows on the order of $O(n^3)$. The maintenance cost per peer is given by

$$d\left(gf_u^{(n)} + Nf_i^{(n)} + gh\left(f_u^{(n)} + \frac{(g-1)f_c^{(i)}}{2}\right)\right) \ , \tag{6}$$

which is of the form $n(k + k'g + k''g(g-1))$ for all peers. This also grows on the order of $O(n^3)$. Consequently, the total cost is on the order of $O(n^3)$.

3) Federated Peer-to-Peer Architecture: For the federated peer-to-peer architecture, the cost per peer is given by

$$h(f_u^{(n)}(g+1) + f_c^{(i)}\frac{(g-1)g}{2}) + f_i^{(i)} \ , \tag{7}$$

the growth of which is on the order of $O(n^2)$. The maintenance cost are given by:

$$dh(f_i^{(n)} + gf_u^{(n)} + f_c^{(i)}g(g-1)/2) \ , \tag{8}$$

which also grows on the order $O(n^2)$. Therefore, the total cost of all peers is on the order of $O(n^3)$.

The cost for all multicast reflectors, including maintenance, is given by

$$Nf_u^{(n)}g(g+1) + dNf_u^{(n)}g + dNf_i^{(n)} \ , \tag{9}$$

which is on the order $O(n^2)$. Thus, the total cost for the federated peer-to-peer architecture is on the order $O(n^3)$.

B. Scalability from a Game Service Provider's Viewpoint

The overall cost evaluation is of less interest for a game service provider that is responsible for the infrastructure of a game and thus maintains servers or multicast reflectors. Clearly, a game service provider is interested in the cost of its equipment only. The peer-to-peer architecture is not relevant for this evaluation as it does not require servers or multicast reflectors. In this section, we compare the infrastructure cost of the client-server and the federated peer-to-peer architectures, that is, the cost of the server and the multicast reflectors.

From Equations (1) and (2), we observe that the cost function of a server takes the form

$$N f_c^{(i)} \frac{g(g-1)}{2} + kn \ . \tag{10}$$

Equation (9) shows the cost for the multicast reflectors.

Both cost functions are second-degree polynomials in g, that is, the total cost is on the order of $O(n^2)$ in both cases. For the client-server architecture, the highest degree factor is $N f_c^{(i)}/2$ and for the multicast reflectors, it is $N f_u^{(n)}$. This means that a central-server architecture is more cost effective if $f_c^{(i)}/2$ is smaller than $f_u^{(n)}$. As $f_c^{(i)}$ denotes the relative processing cost for correlating game states and $f_u^{(n)}$ the relative processing cost for transmitting and receiving state updates, the client-server architecture is preferable if processing state updates is cheaper than transmitting/receiving them. In the case that networking resources are cheap, the federated peer-to-peer architecture is more economical. One important difference is that in the client-server architecture the entire load resides on a single server, whereas the federated peer-to-peer architecture inherently supports multiple multicast reflectors that can be distributed over a network.

C. Scalability from a Player's Viewpoint

The player is interested in reducing the resource cost on its client or peer. Clearly, the best solution in this respect is the client-server architecture, because the resource costs of a client are very small and independent of n.

For the peer-to-peer and the federated peer-to-peer architectures, the cost depends on n. In the peer-to-peer architecture, the dominating term of the cost function is:

$$\frac{g(g-1)}{2} f_c^{(i)} h(1+d), \tag{11}$$

as obtained by adding (3) and (6) or by adding (4) and (6). For the federated peer-to-peer architecture, we add (7) and (8) and obtain the same dominating term. For both architectures, the cost is dominated by $f_c^{(i)}$, that is, by the cost for correlating state updates. The second-order term for the peer-to-peer architecture is $2 f_u^{(n)} h(g-1) + dg f_u^{(n)}(1+h)$ and $h f_u^{(n)}(g+1) + dh f_u^{(n)} h$ for the federated case. These terms represent the network-processing part of the cost; the peer-to-peer architecture is more expensive by a factor of 2 than the federated peer-to-peer architecture. This was expected as a peer in the peer-to-peer architecture transmits state updates to all neighboring peers, whereas in the federated peer-to-peer architecture transmits state updates to a multicast reflector, only.

D. Effect of Partitioning

In this section, we investigate the effect of partitioning the game into several interaction groups. Clearly, increasing the number of interaction groups for the same number of players reduces game-processing cost as less correlation has to be done. On the other hand, the cost of maintaining these interaction groups increases. For the evaluation of all three architectures, we increase N considering the parameters

$n, h, f_i^{(n)}, f_u^{(n)}, f_i^{(i)}, f_c^{(i)}$ fixed. Thus, as N increases, the parameter $g = hn/N$ decreases inversly proportional to N. Note that owing to the intrinsic game logic, the average number g of members in an interaction group cannot drop below a certain limit, i.e., a minimum number of interaction partners is always required. The parameter d, which is the rate at which a peer switches interaction groups, also increases when N increases. The exact relation of d and N is game dependent. However, if we assume a simple model where N represents the number of regions in the virtual game space and we further assume that peers move with a constant linear velocity across regions, then it can be shown that d increases with the square root of N. However, it turns out that in order to compare the different architectures, the exact growth rate of d need not be known, as long as d grows less than N does, i.e. $\lim_{N\to\infty}(d/N) = 0$ and therefore $\lim_{N\to\infty}(dg) = 0$.

1) Client-Server Architecture: For the client-server architecture, the cost for maintaining interaction groups are assumed to be negligible.

The effect of increasing the number of interaction groups is seen in Equation (10), which is dominated by a term of the form Ng^2. As N grows, g shrinks proportionally, i.e., the product Ng is constant. The net effect is that the cost at the server decreases with increasing number of interaction groups. The conclusion for the client-server architecture is to chose as large an N as the game logic permits.

2) Peer-to-Peer Architecture: The game processing cost of a peer in the peer-to-peer architecture is a sum of the form

$$g^2k + gk' + k'', \tag{12}$$

as can be seen in Equations (3) and (4). Increasing the number of interaction groups therefore decreases the game-processing cost.

The maintenance cost per peer, shown in Equation (6), is of the form

$$dNk + dgk' + dg^2k''. \tag{13}$$

The terms containing a factor g decrease as N increases; the maintenance cost is dominated by the term dNk, which increases more than linearly, but less than quadratically, for increasing N.

As a consequence, we conclude that for growing N, the total game cost increases more than linearly. For small N, however, the game processing cost dominates. Therefore, there exists an N for which the total cost is minimum, as shown in Figure 5. Note that this minimum depends on the other parameters, in particular on the number n of players in the game.

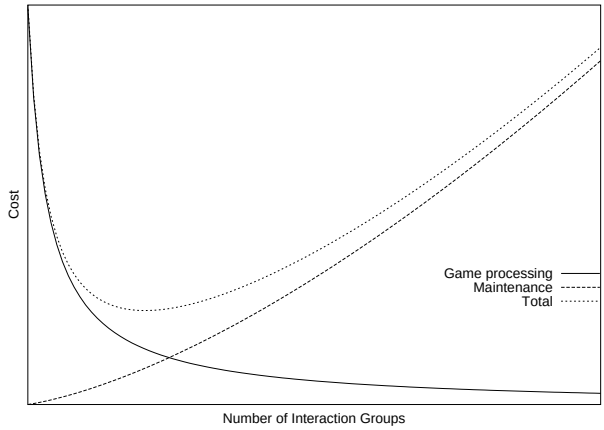

Fig. 5. Game Processing- and Maintenance Cost for increasing N

3) Federated Peer-to-Peer Architecture: In the federated peer-to-peer architecture, the game-processing cost of a peer is of the same form as in the pure peer-to-peer architecture, as seen by comparing Equation (7) with Equations (1) and (2). The cost shrinks as N is increased.

The maintenance cost, describe by Equation (8), is of the form

$$dk + dgk' + dg^2k'' \ , \tag{14}$$

and grows less than linearly as N increases.

The game processing and maintenance costs for the multicast reflectors, shown in Equation (9), also have to be considered. These costs are dominated by the last term, $dNf_i^{(n)}$, which grows more than linearly, but less than quadratically, as N is increased.

This leads to the same conclusion as for the peer-to-peer architecture, namely, that the total cost grows as N grows and that there exists an N for which the total cost is minimum. A game service provider might be interested in finding the minimum cost for all multicast reflectors. We note that Equation (9) has the same general form as the total cost function, and that therefore there exists an N for which the multicast reflector cost is minimum.

E. Local versus Central/Distributed Input Processing

We compare here the cost of local versus the central and distributed input processing for the client-server and the peer-to-peer architectures. To address the scalability issue, we examine the rate of growth as n increases.

For the client-server architecture, for both local and central input processing, the dominant factor of the total server cost is $Nf_c^{(i)}g(g-1)/2$, which implies that the growth is on the order of $O(n^2)$. However, the difference of local and central input processing is linear in n given by $n(f_u^{(n)} - f_i^{(i)} - f_i^{(n)})$. If $f_u^{(n)} > f_i^{(i)} + f_i^{(n)}$, the central input processing approach is more economical than the local one, and vice versa.

For the peer-to-peer architecture, for both local and distributed input processing, the dominant factor of the total peer cost, including the interaction maintenance cost, is $(1+d)f_c^{(i)}hg(g-1)/2$, which implies that the growth is on the order of $O(n^2)$. However, their difference is given by $(g-1)h(f_u^{(n)} - f_i^{(n)} - f_i^{(i)}/g - 1)$, which for $f_u^{(n)} > f_i^{(n)}$ implies that the distributed input processing approach is more economical. This in fact is the case for the game of the Ages of Empire [5].

VI. Summary and Conclusion

Several communication architectures for massive multi-player on-line games exist that aim to support very large games with thousands of players. To the best of our knowledge, none of the earlier works has presented a quantitative assessment of these architectures. In this paper, we have formulated a game model that allows us to assess communication architectures through a set of cost functions. These cost functions quantify the amount of processing and networking resources a given architecture requires. We have applied this model to the client-server, peer-to-peer, and the federated peer-to-peer architectures. The results obtained revealed that, for an increasing number of players, the client-server architecture exhibits the lowest growth in overall system cost, however, with the disadvantage that the entire growth must be handled by the central server. An evaluation of the server components of the client-server and federated peer-to-peer architectures revealed that while both grow quadratically in the number of players, the predominant cost is that of processing game state in the client-server architecture and in the federated peer-to-peer architecture that of the communication cost. Also, the federated peer-to-peer architecture allows us to distribute the resource cost to several components, whereas in the client-server architecture the server is the component that has to cope with the entire load. We have also evaluated the effect of partitioning the game into multiple interaction groups on the different architectures and found that increasing the number of interaction groups reduces the overall cost in the client-server architecture. In the peer-to-peer and federated peer-to-peer architectures, there exists an optimum value for the number of interaction groups that minimizes the overall costs.

The extension of this model in several directions is a subject of future work. This paper has considered the processing and communication resources on server, clients, peers and multicast reflectors. A refined model will also consider networking aspects such as latency and transmission errors. Also, the model presented here assumed linear cost functions and a uniform interaction among the players. In some instances this might not be realistic, such that modified cost functions and more sophisticated interaction modes will then have to be used.

REFERENCES

[1] ANSI/IEEE. *Standard for Information Technology, Protocols for Distributed Interactive Simulation (DIS)*, Mar. 1993.

[2] R. A. Bangun and H. Beadle. Traffic on a Client-Server Based Architecture for Multi-User Networked Games Applications. In *Proceedings of the 1997 International Conference on Telecommunications (ICT '97)*, pages 93–98, Melbourne, Australia, Apr. 1997.

[3] D. Bauer and S. Rooney. The Performance of Software Multicast-Reflector Implementations for Multi-Player Online Games. In *Proc. of the Fifth International Workshop on Networked Group Communications (NGC'03)*, Munich, Germany, Sept. 2003. Accepted for publication.

[4] N. E. Baughman and B. N. Levine. Cheat-proof Playout for Centralized and Distributed Online Games. In *Proc. IEEE Infocom*, pages 104–113, 2001.

[5] P. Bettner and M. Terrano. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. In *The 2001 Game Developer Conference Proceedings*, San Jose, CA, Mar. 2001.

[6] C. Diot and L. Gautier. A Distributed Architecture for Multiplayer Interactive Applications on the Internet. *IEEE Networks magazine*, 13(4):6–15, July/August 1999.

[7] B. N. Levine, J. Crowcroft, C. Diot, J. Garcia-Luna-Aceves, and J. F. Kurose. Consideration of Receiver Interest for IP Multicast Delivery. In *Proc. IEEE Infocom*, volume 2, pages 470–479, 2000.

[8] D. Levine, B. Whitebook, and M. C. Wirt. *A Massively Multiplayer Manifesto*. Butterfly.net, Inc., 123 East German St. Shepherdstown WV 25554, May 2002. Version 1.1.

[9] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. Exploiting Reality With Multicast Groups: A Network Architecture for Larce-scale Virtual Environments. *IEEE Computer Graphics and Applications*, 15(5):38–45, Sept. 1995.

[10] OpenSkies. OpenSkies Network Architecture. http://www.openskies.net/papers/papers.html.

[11] S. Rooney, D. Bauer, and P. Scotton. Efficient Programmable Middleboxes for Scaling Large Distributed Applications. In *6th International Conference on Open Architectures and Network Programming (OPENARCH)*. IEEE, Apr. 2003.

[12] J. Smed, T. Kaukoranta, and H. Hakonen. A Review on Networking and Multiplayer Computer Games. Technical Report 454, Turku Centre for Computer Science, Apr. 2002.

[13] Sony Online Entertainment Inc. EverQuest. http://www.everquest.com, 2002.

[14] L. Zou, M. H. Ammar, and C. Diot. An Evaluation of Grouping Techniques for State Dissemination in Networked Multi-User Games. In *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Cincinnati, Ohio, 2001.