VFC-RTS

Aplicação do modelo VFC aos jogos de estratégia em tempo-real

Manuel Eduardo Costa Cajada

INESC-ID/Instituto Superior Técnico
Distributed Systems Group
cajadas@gmail.com

Abstract. Although MMOGs have been gaining most popularity over the years, RTS games have not been considerate a strong candidate for using this model because of the limited number of players supported for the usually employed network topologies, large number of game entities and strong consistency requirements. To deal with this situation, the notion of location-awareness can be extremely useful to confine the zones with consistency requirements. Existing interest managing techniques present visibility loses and intermittent object presence near the player's area-of-interest, creating a confusing and annoying experience for the user. In this work we propose VFC-RTS, an adaptation of the VFC model, characterized for establishing consistency degrees, to the RTS scenario. This is especially relevant since in RTS games, game scenarios (e.g., battlefields) tend to be very large, populated with large number of game entities (soldiers, tanks, etc.) and players are engaged in multiple locations (e.g., battlefronts). We start by making an overview about the state-of-the-art in the fields of consistency maintenance, interest management and RTS games. Then we present our solution for the VFC-RTS system and propose an architecture for a generic middleware. Finally, we present our evaluation method for the implementation of the middleware.

Keywords: real-time strategy, massive multiplayer online games, consistency, VFC.

1 Introduction

With the increase of the internet services capabilities over the years (i.e. larger bandwidth and packet throughput) there has also been a growth in the number and variety of applications. Among these new types of real-time applications, one that has been gaining most popularity are massive multiplayer online games (MMOG). Games of this class distinguish themselves from other kinds of online games by supporting a great number of players sharing a virtual world, possibly very large, while interacting with each other and with the surrounding environment.

In the world of the MMOGs, the category most played and with the greatest popularity is that of MMORPG. Typical examples are titles like *EverQuest*[1] and *World Of Warcraft*[2]. In these games, the participants play the role of a character in a virtual world, experiencing it through the eyes of an avatar. There are other real-time network game categories which have been less discussed for using the MMOGs model, among them the real-time strategy games (RTS)[20]. In this class of games, the player manages a set of entities, usually varying between different kinds of structures and pawns,

[1] http://everquest.station.sony.com
[2] http://www.worldofwarcraft.com

with the objective of commanding the troops throughout the map until achieving total control over it.

Multiplayer solutions for RTS games typically allow a limited number of participants to be playing simultaneously (in the order of dozens)[7] due to the lack of scalability of the traditionally used network architecture, the Client-Server(CS) model. This architecture is characterized by the use of a centralized server that is responsible for maintaining the game state by collecting updates from all clients, resolving inconsistencies and propagating the most current game state. Other network models have already been considered to deal with the scalability issue, among them the Peer-to-Peer (P2P) model[12,13]. In a P2P architecture, each peer is both a client, responsible for executing its own game simulation[26], and a server for other peer clients. Although this results in decentralized resource consumption and an increase of game resources every time a node joins the system, it is most costly in bandwidth consumption because it takes more messages to synchronize game state between all participants, resolve inconsistencies and coordinate overall game progress. Hybrid solutions have also been presented that balance the trade-off between performance and communication costs, like Peer-to-Peer with Central Arbiter (PP-CA)[26] and Proxy-Server[20].

When playing over the internet, users are subjected to internet service conditions that may not always satisfy the game's QoS (Quality of Service) specification. Latency is often the greatest network related issue because it affects the usability of the games, compromising users QoE (Quality of Experience). On a FPS (First-person shooter) the latency value generally tolerated has to be less than 50 milliseconds[6]. However, for a RTS, the latency can achieve higher values (up to 200 milliseconds) without affecting the game.

"Latencies up to several seconds have little effect on the final outcomes of building, exploration, and most combat. Overall, strategy plays a much larger role in determining the outcome of the game than does latency" [7]

Maintaining game state consistency on a RTS game is a complex task. Not only does a player have to be aware of the progress of his troops throughout the map (considering troops in the order of hundreds or even thousands), but also maps on a MMOG tend to have very large dimensions. In addition, the system has to be prepared to deal with changes in network quality without losing performance. To reduce the amount of data on a state update process and still provide the user with the relevant state information, MMOGs employ *information management*(IM) techniques.[5,39]. This concept, commonly used on other MMOG classes of games like RPG[2,5] and FPS[18], restricts the amount of data messages during the update process by limiting the area of interest (AoI) of a player. Thus, the player only receives the game state data necessary to notice changes on the scenery up to a certain range, mostly corresponding to the area where avatars and events relevant to him are located and take place. In spite of reducing the bandwidth consumption, existing IM techniques present an all-or-nothing approach, providing the user with every update about objects inside his AoI and non about objects outside. This situation can result on visibility losses, intermittent presence and movement of objects near the AoI boundaries, and decrease on playability.

There are techniques to deal with changes in network conditions without compromising the consistency. *Dead reckoning* for instance, reduces the bandwidth usage by sending update packets less frequently. By using a prediction algorithm the system can compensate information gaps, avoiding an inconsistent state. Other techniques like packet compression or aggregation are used to minimize network traffic and bandwidth requirements respectively[31].

This paper proposes adapting an consistency model design for games to the needs of the RTS game class. By doing this it is expected to notice an increase of game scalability which can be translated in the increase of the maximum number of player on a single session, game entities controlled by the player or even of the maximum number of units on map, without compromising game performance or the player's QoE. The model implemented is the *Vector-field Consistency*(VFC) model[28,37]. Designed for the gaming environment, VFC presents its own IM technique, characterized by the use of *consistency zones*, presenting increasing consistency requirements as the distance from the pivot decreases. The goal of this paper is to adapt the IM algorithm proposed by the model to the world of RTS games. In order to maximize the number of applications that can make use of this solution, it will be implemented as a *middleware*, separating the game's logic from the consistency model implied. *Warzone 2100*[3] was the game chosen to test the solution. *Warzone 2100* is an open-source militaristic RTS game with a fully 3D environment and supported by most of the operating systems.

To evaluate the solution and determining if it improves the scalability of the game it is used the Game Scalability Model[19,20]. Not only the GSM allows an analytical study about the scalability levels achieved by the network topology but also, when incorporated in the game, self-monitoring of resources utilization at runtime.

The rest of this paper is organized as follows: Section 2 presents the objectives of this work, followed by Section 3 where we explore the topics of consistency maintenance, existing solutions and how they apply to the MMOG scenario. Section 4 proposes a solution for the described problem. Section 5 details the parameters that will be used to analyze the success of the solution. We conclude the paper in Section 6.

2 Goals

The development of the VFC-RTS pursues four major goals:

1. **Adaptation and optimization of the VFC model to the RTS environment:** Modifying the premises of the Vector-field Consistency model in order to adapt it to the reality of the RTS games. In particular, the main goal is to improve the system scalability without compromising availability, performance or playability.

2. **Study, analysis and evaluation of the state-of-the-art:** Researching the topics of consistency models, IM techniques and MMOGs, including network architectures. Examination of the existing solutions, how they relate, their advantages and disadvantages.

3. **Implementation of the adapted algorithm:** Concretization of the solution as a *middleware* and its application to a commercial game.

4. **Incorporation of the GSM and evaluation of the solution:** Usage of the Game Scalability Model to measure the execution time of several tasks that have to be accomplished during a running game session. The measurements are then used to calculate a forecast of the maximum number of players supported. Comparison of the values provided by model before and after the VFC solution.

3 Related Work

The following section addresses a study about three major topics to determine the state-of-the-art of Consistency in Distributed Systems focusing the RTS games environment.

[3] http://wz2100.net

First, an overview on the general consistency scenario, exploring the different design choices and kinds of systems (section 3.1). Second, we present how general consistency concepts apply to the game environment (section 3.2). Finally, in section 3.3, we present the different architectural solutions for MMOGs, focusing on RTS games.

3.1 Consistency in Replicated Systems

In the field of Distributed Systems, high availability and performance are two major requirements assured by the use of data replication. By maintaining multiple replicas on different computers, a system can assure that the failure of one or more replicas will not cause the data to became unavailable. Furthermore, by allowing each user to connect to the nearest replica to access data, data replication also improves load balancing, thus improving performance. The issue of maintaining consistency in distributed systems can be described as the "ability to keep replicas sufficiently similar to one another despite operations being submitted independently at different sites"[27]. However, as there are multiple replicas there is also the possibility of information disparity between them, leading to inconsistency. In the next sections we will present two distinct classes of replication techniques: *pessimistic replication* (section 3.1.1) and *optimistic replication* (section 3.1.2), followed by a review of some well stablished systems that explore the consistency spectrum (section 3.1.3), and finally, in section 3.1.4, we presented an overview about the discussed topics, making an comparison between the presented solutions.

3.1.1 Pessimistic Replication

Pessimistic replication techniques aim at maintaining single-copy consistency, usually by usage of locking mechanisms[21]. Although users can read data from any of the existing replicas, as long as it is up to date, only one user at a time is allowed to perform write operations to a replica, blocking the remaining users during the process. These provide strong consistency by preventing conflicts between updates since data changes are isolated and propagated to all the other replicas, becoming available for all the following read operations. Despite being widely used in commercial systems[14] due to performing well in local-area networks, this solution is not suited for the Internet, as intermittent connectivity could result on blockage of the system or even data corruption. In addition, pessimistic replication algorithms scale poorly because of the frequent update propagation, compromising performance and availability.

3.1.2 Optimistic Replication

Contrary to the pessimistic approach, optimistic replication algorithms[27] are based on the assumption that conflicts between updates occur only rarely and can be easily fixed when they happen. As a result, optimistic algorithms allow replica data accesses without previous synchronization between all replicas. Hence, less frequent communication is required, leading to better performance and availability. Additionally, optimistic replication increases flexibility, scalability and enables asynchronous collaboration, making it the ideal solution for wide-area or mobile environments[27]. Optimistic replication introduces the concept of *eventual consistency* which states that even though the state of replicas may diverge, the continuous propagation of updates in background will eventually lead to the full consistency of the system. While in pessimistic consistency the sequence of operations to an object in a site must be preserved when applied on the remaining sites, optimistic replication can support partially inconsistent data. Nevertheless, this property makes this model unsuitable for systems with strict consistency requirements that cannot tolerate occasional conflicts.

The design of an optimistic consistency system demands a number of choices regarding the properties expected from the application. The following portion of the paper presents an overview of these aspects and how they influence the outcoming system.

Number of writers. An optimistic system can be classified according to the number of writers, as **single-master**, where one replica is assigned the master and all updates are submitted by it, being then propagated to the remaining replicas, or **multi-master**, letting updates be submitted by any replica independently. Even though multi-master systems provide greater availability, they are much more complex as they have to be able to handle conflicts and scheduling, in opposition to single-master systems.

Definition of operations. This property takes into consideration how updates are disseminated from the replica where they originated to the other replicas. This process can be done by **state-transfer**, substituting an entire object every time a byte is modified, or **operation-transfer**, transferring the operations corresponding to the user changes, recreating the changes to the objects on the other sites. State-transfer[28] is a simpler procedure, compared to operation-transfer, and can result in easier convergence of replicas in cases where the application of every missing update proves to be more costly than replacing the entire object. This proves suitable for situations when sites stay off-line for long periods of time. Although operation-transfer[29] is a more complex approach, since it requires maintaining the history of operations, it proves to be more efficient when objects are large and operations high-level, and more flexible for conflict resolution, reducing bandwidth requirements.

Scheduling. In order to achieve consistency, every replica has to produce the same schedule of event, resulting in a consensual final state. There are two different scheduling policies: **syntactic scheduling** which orders operations based on absolute information such as submission timing, location and issuer, and **semantic scheduling** which makes an effort to organize operations in an "optimal" way, exploiting semantic properties like commutativity with the goal of reducing conflict occurrence and rollbacks. Syntactic scheduling proves to be a simpler solution since it requires no decision making, disregarding relations between operations. However, this may cause the detection of unnecessary (or even false) conflicts, avoidable by such things as changing the order of execution of two concurrent operations, only possible with semantic scheduling.

Conflict Handling. One of the defining characteristics of a distributed system is how it handles the occurrence of conflicts between updates. Conflicts are detected when operations do not satisfy their prerequisites. There are several way to manage conflicts and they all are divided in two phases: **detection** and **resolution**. Pessimistic algorithms and single-master systems handle conflicts by preventing them from happening, restricting the number of users capable of submitting updates, one at a time in the former, and assigning one replica as the one writer of the system in the latter. This solution, however, affects negatively the availability of the system. Other systems handle conflicts by ignoring them, leading to an increase of lost updates and, consequentially, of difficulties in data management. Conflict detection policies, like scheduling policies, are classified as **syntactic** or **semantic**. Syntactic conflict policies do not present explicit prerequisite specification, making use of happen-before relationship [15] to intuitively capture the relations between operations. Semantic conflict policies take advantage of semantic information about the operations to designate if a conflict has happened. Although syntactic conflict policies are more efficient than semantic policies they lack flexibility and incur in a higher number of false conflicts, due to the assumption of the happen-before relationship that any two concurrent operations are conflicting.

Regarding conflict resolution, there are two practices: **manual**, presenting to the user the two conflicting versions and letting him decide which will be the final one, or **automatic**, used in such systems as Bayou[35], which requires the definition of pre-conditions and merge procedures to each operation. Although this approach is most appealing and some times easily implemented (using time references or priority systems), studies prove that writing merge procedures can also be a very complex task.

Propagation Strategies and Topologies. Update propagation is a key element of an optimistic system, defining how and when updates are distributed among replicas. Propagation strategies can be categorized in terms of **communication topology** and **degree of synchrony**. Topologies with a fixed structure can be highly efficient but lack the dynamism needed for failure-prone and unstructured networks. These kinds of networks rely on epidemic propagation algorithms, allowing operation propagation even in networks with dynamic structure.

The degree of synchrony is a unit of measurement for the frequency and speed of inter-site communication and update exchange. The range of the degree of synchrony goes from pull-based systems, where a replica has to request updates from the remaining replicas, to push-based systems, where a replica with new updates proactively informs other replicas. To obtain the lowest degree of replica inconsistency and rate of conflict, the general approach is to apply quick update propagation, compromising the complexity of the system.

Consistency guarantees. As the state of replicas may diverge, there are consistency guarantees that stipulate how much the state between replicas may differ. In the spectrum of allowed divergence there are two extremes: **single-copy consistency**, ensuring that the result of multiple writes from different sites is the same as a serialized execution of operations in a single site, and **eventual consistency** that guarantees only that the state of replicas will eventually converge. This concept offers weak assurance about when consistency among replicas will be reached, however, it is the common choice for optimistic-replication systems, designed to accomplish availability. Hybrid consistency models have been proposed which seek solutions between single-copy consistency and eventual consistency, often achieved by blocking access when consistency conditions are not met (divergence bounding)[40,28] (section 3.1.3).

3.1.3 Case-study Systems

In this section we present a review of four case-studies, representing the evolution of the optimistic replication scenario. First, a mature technique by the name of *Operational Transformation* that applies the semantic scheduling schema, followed by three divergence bounding systems: *TACT*, *VFC*, and finally, *Data-aware Connectivity*.

Operational Transformation. Designed for real-time cooperative editing systems, operation transformation (OT) is a consistency maintenance technique that makes use of semantic scheduling.

The first system to implement the concept was the GROVE[10,32] system, which used a replicated architecture, where each participant would have a local replica of a shared document and when a participant wanted to submit an update he would execute it immediately on the local copy and then propagate the update to remotes sites for execution. Before executing the update, the remote replicas apply a transformation to the operation, discarding the need to reorder previous operations. The GROVE system identified two major inconsistency problems regarding the execution of remote operations: *divergence* and *causality violation*. The problem of divergence emerges from the fact that, if a set of non commutative operations is not executed in the same order

in every site, the execution will result in different final states. The causality violation problem can be easily detected in a situation where replicas execute received updates immediately after arrival, without respecting natural causal order.

With the previously announced problems as its basis, GROVE defines two properties as consistency correction criteria: **convergence property**, stating all generated operations have to be executed at all sites, and **precedence property**, stating if one operation a causally precedes another operation b, then at each site the execution of a happens before the execution of b. The solution presented by GROVE consists of two components: a *state-vector timestamping*[25] scheme to implement the precedence property and to ensure the convergence property the *distributed OPerational Transformation* (dOPT) algorithm. The dOPT algorithm performs a transformation to every operation that complies to the precedence condition against independent operations. The outcome of the transformation should be a set of operations that will produce the same final state when executed in any site.

More than a decade later, REDUCE[34] explored a third inconsistency problem designated *intention-violation problem*, related to the execution of a sequence of operations not producing the results expected by the user even though replicas are consistent. To deal with this problem, the authors of REDUCE defined the *CCI Consistency model*, stating that consistence is only achieved when the previous presented GROVE properties are contemplated plus the *intention-preservation* property: for any operation O, the effects of executing O at all sites are the same as the intention of O, while not changing the effect of independent operations. To ensure these properties, REDUCE applies a distinct approach to achieve convergence: an undo/do/redo schema and implements the *Generic Operational Transformation* (GOT) algorithm[33] to enforce intention-preservation, making use of pre and post conditions to capture the user intention when transforming operations.

GROVE was a pioneer in the field of operational transformation, opening doors for new systems such as Jupiter[23], which ported the operational transformation paradigm to the scenario of shared persistent virtual worlds.

TACT. In the spectrum of consistency, designers of replicated systems are normally forced to choose between one of the two extremes, pessimistic replication, associated with performance overheads and limited availability, and optimistic replication, characterized by eventual consistency. The TACT (Tunable Availability and Consistency Tradeoffs) framework[40] is a continuous consistency model that enables the designer to bound the consistency requirements of the application. Thus, it allows the designer to define the maximum semantic distance between replicas and even different consistency boundaries per-replica. When the operation meets the pre-established requirements it proceeds locally, otherwise it blocks the operation until synchronizing with other replicas as specified by the system's consistency requirements.

On TACT, an object consistency semantics are defined by *conits*, using three metrics: (1) *Numerical error*, which bounds the consistency distance between the local value of the conit and the value of the "final image", i.e an image of the system where all updates have been applied, (2) *Order error*, used to quantify the difference between updates order on the local replica and the ordering on the "final image" and (3) *Staleness* is the maximum delay allow between the current time and the acceptance time of the oldest write on a conit not seen locally. The implementation of the first metric is easily accomplished for applications with numerically identified operations, needing to specify the weight for each write operation otherwise. The numerical error is then the weight of all writes applied to a conit at all replicas unseen for the local replica. This value can be hard to obtain as it relies on the cooperation between all replicas, implying the use of a consensus algorithm to update it. Order error bounds the maximum number of writes that can be rolled back or reordered, making use of a write commitment

algorithm and compulsory write pull to implement the order error bound. Staleness is maintained using a real-time vector that holds a entry for each replica stating the time passed since the last update.

TACT contributed to the consistency scenario by presenting a solution where different consistency levels can be assigned to different replicas. A replica allocated on a mobile device, for instance, which is subjected to poor network conditions and processing power constrains, may relax its consistency to reduce the incoming communication amount (updates generated on other sites), without compromising the consistency of the remaining replicas (outgoing updates remain unchanged). In the same way, a replica with great communication conditions and processing capability may impose strong consistency at little cost. The TACT framework take advantage of this property to increase the performance of the system by routing clients to replicas that meet their consistency requirements.

VFC. The Vector-Field Consistency model[28,37] is, as the TACT framework, a continuous consistency solution that allows a per-replica consistency level specification. However, contrary to the TACT solution, it also allows to change consistency requirements dynamically based on the application's run-time state. Moreover, it applies the concept of *locality-awareness* to increase availability, user experience and reduce resource consumption. To implement *locality-awareness* it is necessary to establish one or more points (sets of coordinates specified by the developer, or dynamically by the application) from which is calculated the distance to the surrounding objects. This points are called the *pivot points*. A pivot point represents the position around which there are strong consistency requirements, weakening as the distance from the pivot increases. In order to define the consistency bounds, delimitating the authorized divergence degree, with the pivot as the center, there are drawn ring-shaped, concentric areas presenting increasing radius and divergence degree, called *consistency zones*. This design concept is further explored in section 3.2.2.

To formalize the consistency degree of a zone, VFC employes a 3-dimensional consistency vector: (1) *time*, the maximum time a replica can be without the last update values (2) *sequence*, the maximum number of lost updates supported, and (3) *value*, the maximum divergence between replicas or against a constant. The values chosen for the vector should produce a consistency guarantee so that it does not compromise the user experience by presenting bad information.

VFC proves to be an very flexible solution as it allows explicit and dynamic definition of consistency degrees, suitable for a wide variety of systems. Furthermore, it is developer-friendly thanks to its simple and intuitive settings parameterization schema, and player-friendly since it guarantees the player is provided with all the information needed to make sensible decisions. Also, VFC accomplishes an efficient management of the system resources by, intelligently, classifying updates with different priorities, sending critical updates immediately and postponing less-critical ones. Designed originally for the distributed games environment, the VFC model can easily be ported to other environments by adapting the concepts of pivot, *consistency zones* and distance to the target systems. One area where it has presented successful results is on the field of cooperative work tools[8].

Data-aware Connectivity. Intended for the mobile environment, the goal of the Data-aware Connectivity[3] (DaC) system rests on the principle of ensuring acceptable quality of a replicated object at minimal connectivity resources cost. To reach this goal, the system implements a continuous consistency model that enforces data consistency bounds of local replicas, along with a *connectivity monitor* that, using knowledge about the existing communication supports, calculates the less expensive connection to obtain the desired quality levels. When the system cannot exploit any connection

successfully it refuses accesses to the replica. This solution proves to be greatly useful for mobile systems, where, nowadays, a larger number of network technologies are available, presenting different price costs, ranges and resources consumption levels, by intelligently choosing the best suited connection for the given task. Additionally, by leaving decisions regarding object refreshing and connection choosing to the system, not only it becomes more transparent to the user but also reduces error-prone user decisions.

The Data-aware Connectivity is composed of two modules: a *quality monitor* and a *connectivity regulator*. The quality monitor is responsible for continuously estimating the divergence degree of each replicated object. To calculate when to refresh an object, the quality monitor considers, among other criteria: (1) time since the last synchronization from each replica; (2) number of local tentative updates; (3) commit weight of currently missing from accessible replicas; (4) number of known concurrent updates; and (5) recent update activity by other replicas to the object. By contemplating these criteria, the quality monitor ensures the properties of freshness (1), consistency (2) and possibility of quick commitment (3-5).

Existing continuous consistency systems, such as TACT or VFC, ensure replica quality by forbidding access to replicas once they exceed the allowed divergence degree and until they are obeyed again. However, in this system when replica quality is below the bounds defined, it probes available connectivity in order to reestablish consistency levels, forbidding replica access only if no connection is possible.

3.1.4 Conclusions

From the design standards previously presented some conclusions can be taken regarding the kind of optimistic system intended. Single-master systems are well suited for applications with multiple readers or a single writer and with little conflict tolerance. Multi-master systems prove to be a better option for applications with multiple writers but rely on eventual consistency. State-transfer systems are simple to manage and simple to implement, presenting low memory and constant bandwidth requirements, what makes them adequate for most optimistic systems. Nevertheless, the fact that these systems cannot resolve conflicts between operations, as do operation-transfer systems, makes them unsuitable for applications with an elevated number of concurrent events, and for applications that may produce small modifications to files of big dimensions, compromising the network's bandwidth.

	Systems			
	OT	TACT	VFC	DaC
Number of Writers	multi-master	multi-master	multi-master	multi-master
Dissemination	operation	operation	operation	state/operation
Topology	any	any	centralized	any
Degree of Synchrony	push	push/pull	push/pull	push/pull
Server Update Sending Order	when ready	when ready	priority	priority
Update Execution Order	reception	reception	reception	reception
Consistency Guarantees	eventual	bounded	eventual/bounded	bounded
CPU Load	high	high	high	medium
BW Load	low	low	low	very low
Mobile Support	no	no	yes	yes
Divergence Bounding	no	yes	yes	yes
Locality Awareness	no	no	yes	no

Table 1: Optimistic replication system's design choices

Table 1 summarizes the section by presenting the relevant optimistic replication design choices taken in the case-study solutions. By analyzing the table it is possible to point out the main differences between the systems:

(1) While OT and TACT are suited to any network topology, VFC, designed originally for small wireless network games, implies the use of a centralized architecture. More recent work has successfully applied the VFC model to a distributed architecture[37,22].

(2) Contrary to the OT and TACT ordering solutions, where updates are sent as they are generated, VFC uses a priority system, sending critical operations first than less critical ones. Although this system implies the classification of each operation regarding its context and content importance, increasing the computational load, it is a small compromise compared to the reduction of bandwidth consumption.

(3) Regarding consistency guarantees, each system takes a different position: Although OT offers weak consistency guarantees and TACT establishes consistency boundaries by specifying the divergence degree allowed by the application, VFC presents a hybrid solution in which the area closer to a pivot point presents stronger consistency and as the distance from the pivot increases the consistency requirements go weaker, leading to, ultimately if desired, just eventual consistency.

3.2 Consistency in Games

One of the aims of MMOGs is to provide an environment where realistic interaction among players can occur. For that, every participant should be unaware of any data inconsistency regarding entities of the game. In this section, we present how the replicated distribution paradigm previously analyzed is applied to MMOGs reality (section 3.2.1), techniques to, in real-time, identify the AoI of the user (section 3.2.2), and techniques to minimize the network communication load, such as *Dead Reckoning* (section 3.2.3), packet compression and aggregation (section 3.2.4).

3.2.1 Adaptation and Usage of Classic Consistency Approaches

In this section of the paper we present how the different approaches for consistency management previously presented are used in the distributed multiplayer games domain.

Pessimistic Approaches. In order to maintain game state consistency in each participant, pessimistic algorithms make use of lock or flow control methods[21]. Lock based approaches are similar to *token ring* type network which allows data transmission for a node that has the token in the network. A user is allowed to manipulate a shared object exclusively when he is in possession of the token, sending the token to the next element of the ring afterwards, possibly with the updates piggybacked in the token message. Even though these approaches provide strong consistency and are simple to implement, transferring the ownership of the token among the nodes can be delayed due to network latency[24], compromising the user experience.

Optimistic Approaches. Optimistic consistency algorithms intend to maximize real-time interactivity among participants by removing the necessity of waiting for the ownership of shared objects. To accomplish this goal, the local game state is immediately updated and then transmitted to other participants by state or operation transfer. Although these approaches imply immediate responsiveness even in poor network conditions, when inconsistencies arise, state repair procedures are in order, which

may increase the computational load and reduce playability. *Frequent State Regeneration*, *Bucket Synchronization* and *Time Warp* belong to the optimistic consistency maintenance approaches.

Frequent State Regeneration[30] is a state-transfer algorithm that propagates updates by regularly sending packets containing complete data regarding current frames in game sessions, disregarding the occurrence of state changes. Thus, the receiver can maintain consistency by identifying and repairing the divergence between the local state and the remote state. Games employing this algorithm include titles like *Doom*[4] and *Diablo*[5]. This algorithm is simple to implement as it does not require central servers or lock management. Nevertheless, it scales poorly, as the number of objects in a session increase, and presents elevated bandwidth requirements.

The Bucket Synchronization mechanism, employed in the game *MiMaze*[16,31], divides the game time in blocks of the same length and each block presents a "bucket", a set of updates. Remote updates, as well as local updates, are assigned to the appropriate bucket according to the prefixed playout delay. This delay determines how much time must be passed to display the results of user actions. In this way, a message issued at absolute time is delayed according the playout delay so that all participants can evaluate it at the same time. The data stored in the buckets are used to render game frames in sequential order. In case a message is missed or arrives too late, the algorithm uses data from the preceding bucket to extrapolate the information, using *Dead Reckoning* (section 3.2.3).

The Time Warp[38] algorithm adopts the virtual time paradigm, characterized by the use of a 1-dimension temporary coordination system for distributed processing management, exploiting the logical clock conditions proposed by Lamport[15]. On a normal execution, the algorithm processes any update (local or remote) that presents a timestamp (virtual time) larger than the last processed event, leading to a global ordering of events. However, if an arrived update presents a smaller timestamp than the latest one, a rollback procedure is triggered, canceling all events referring to updates not correctly ordered and processing the reordered queue of events. To cancel incorrectly ordered events, Time Warp sends an anti-message for each message previously flagged, which will cause the rollback of other processes. This procedure may cause degradation of the system's performance as well as flooding of the network with anti-messages.

Other consistency models, such as the continuous consistency model proposed in [17], approach specific areas of state consistency, in this case, object trajectory. This particular model proposes a *relaxed consistency* approach that relaxes the strict time-dependent consistency control on individual states of a replicated object, allowing the object to deviate its trajectory by an acceptable amount.

3.2.2 Interest Management

In order to provide a shared sense of space among players, each player must maintain a copy of the **relevant** game state on his computer. The simplest approach requires every player broadcasting updates to all other players so that each player can maintain a full copy of the game state. However, this solution is proved to scale poorly. IM techniques[2,5] reduce the number of updates to be transferred to other players by utilizing filtering mechanisms such as application specific and/or network attribute specific. In this section we focus on application specific IM politics and classify a number of IM algorithms according to them.

[4] http://www.idsoftware.com/games/doom
[5] http://eu.blizzard.com

Interest Management schemes are usually classified, according to the range of action, as based in: *aura-nimbus*, *range* or *visibility*.

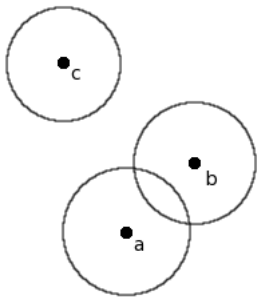

Fig. 1. Example of *aura-nimbus* IM

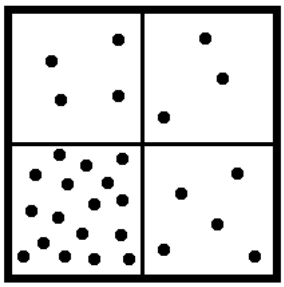

Fig. 2. Example of region-based IM

Aura-Nimbus. The aura is the area that bounds the presence of an object in space while the nimbus corresponds to the area-of-interest of an object, the area in which an object presence can be perceived by the others[5]. In its simplest model both the aura and nimbus can be represented by fixed-size circles around the object (Fig.1). Considering an observer object a and observable object b, when the aura and nimbus are coincided with each other, a can receive the information in regard to b. This model proves to be advantageous by allowing a fine-grained interest management, limiting the number of sent updates to only the relevant ones. However, it scales poorly due to the cost of computation associated with the identification of intersections between auras and nimbus.

Range. Used in such systems as RING[11] to mitigate the limitations of the aura-nimbus model, range-based interest management partitions the world space into static regions, assigning each object on a region to the AoI of every other object it the same region (Fig. 2). This model is often cheaper to compute than a pure aura-nimbus model. Nevertheless, the quality of the model is dependent of the shape and size of the regions along with the distribution of object among the regions. In a situation when several objects are concentrated in one region (illustrated in Fig. 2), load balancing mechanisms are often required. Although regular square tilings are the most popular choice, hexagons make a better approximation to the aura-nimbus model.

Visibility. Contrary to the other IM approaches presented, visibility-based interest management considers the vision of the object/player instead of a fixed radius, taking in great consideration the geography of the game environment. The RING system, referred in the last model for applying a region-based approach, also makes use of the visibility model for precomputing visibility between regions[11], limiting the updates received by an object not only to its region but also to the range of its vision. The advantage of this technique is that the visibility of the objects is determined precisely and without additional cost since object information is already computed for rendering. However, this model assumes that a client has already received the information regarding all objects that may be visible in the world, or, in systems like RING that

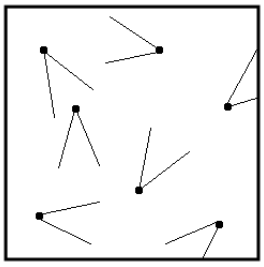

Fig. 3. Example of visibility-based IM

also apply region-based IM, visible in its region.

To obtain the finest-grained update filtering, many systems opt to apply more than one IM model. One example of this is the already referred **RING** system. Designed to support a large number of players in a shared environment, the RING system applies both a range-based model, limiting the AoI of every object to the region they are positioned, and a visibility-range model, further restricting the interest zone to the area the player's visibility can reach.

A3[4], an algorithm intended for the MMOGs environment, combines the aura-nimbus and the visibility-based models in order to limit the AoI of the player to an area of 180 degrees in front of the *avatar*, ignoring objects localized in the back. This property may affect the user experience in situations when the *avatar* makes sudden rotation movements, causing a delay on the objects to become visible. To handle this situation in a less brusque manner, the algorithm also considers a second aura of smaller dimensions, retrieving information about only the closest objects behind the avatar. Moreover, A3 also includes a notion of *consistency degree*, requesting more often updates to objects closer to the *avatar* and decreasing the frequency of updates as the distance from the *avatar* increases.

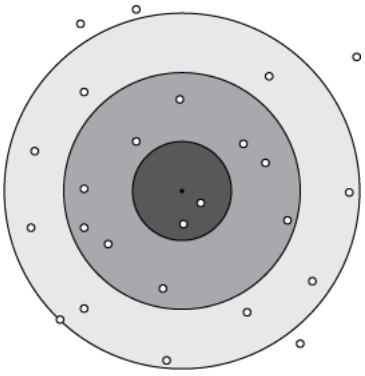

Fig. 4. 2-dimensional VFC

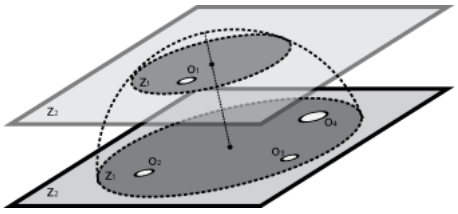

Fig. 5. 3-dimensional VFC

VFC[28,37], presented in section 3.1.3, applies multiple concentric aura each with different radius. This design complements the VFC consistency model, limiting *consistency zones* as rings around a *pivot* point (Fig. 4). This way objects situated in rings

closer to the *pivot* point present greater consistency requirements which decline as the rings go wider. Moreover, the VFC interest management technique is also supported in a 3-dimensional environment (Fig. 5). This can be accomplished by porting the shape of the AoI to the 3D spectrum (concentric spheres). In the solution, objects situated within *consistency zones* on top of the *pivot* point present the same consistency requirements any other object in the same zone.

3.2.3 Dead Reckoning

In opposition to the application specific solutions previously discussed to reduce the bandwidth use, network specific approaches, such as Dead Reckoning[31], seek this goal by sending update packets less frequently. Especially effective when handling position information, Dead Reckoning makes use of approximation techniques to extrapolate the information of missing updates based on updates previously received (therefore leveraging semantic knowledge of the game and to some extent, location-awareness). Additionally, Dead Reckoning applies convergence algorithms to avoid abrupt movements when new location information is applied.

To achieve the most accurate approximation results, prediction techniques may employ such trajectory parameters as velocity and acceleration. The values of these variables can be obtained in two different ways: approximated from the history of last received updates or attached in each object's position update. Although the second way is more accurate, the communication overhead associated to the transmission of additional information leads to an increase of bandwidth consumption. Predicting the velocity and acceleration of an object proves to be less costly bandwidth wise, however, if new location information is not received through large periods of time, small inaccuracies in the algorithm may accumulate in significant errors.

When a node receives an update message, the object's predicted position is likely to differ from the position based on the latest information. To deal with this situation, Dead Reckoning makes use of convergence algorithms that vary in complexity. The simpler solutions, such as moving the object directly to the position on the latest update, have no additional computation requirements but produce jerky movements. More complex algorithms pick additional points between the two positions to produce a smoother movement but have greater computation requirements.

3.2.4 Packet Compression and Aggregation

Packet compression techniques can reduce average packet size by encoding packet header and/or body to be transferred to other nodes[31,9]. In general, compression methods can be divided into two groups, ***lossless*** and ***lossy*** compressions. On the one hand, lossless compresions methods can restore original data from encoded one without any loss, usually capable of shrinking the size of data to approximately down to half. On the other hand, lossy compression algorithms are not required to recover the original state of compressed data. These algorithms are broadly utilized to compress multimedia data significantly compared to the lossless compression by exploiting the imperfection of human perceptibility such as visibility and audibility.

Packet compression methods can be categorized into **internal** and **external** types. Internal compression algorithms encode packets without referring previously transferred packets. This types is advantageous for the easiness of implementation and no requirement of reliable protocols, however, it presents a low compression rate. External compression methods can achieve high compression rate by utilizing packets transmitted beforehand but require storing of packet history. Furthermore, reliable transmission protocols are demanded to uphold the dependency of packets.

The bandwidth requirement can be alleviated by utilizing packet aggregation techniques. Aggregation techniques reduce the overhead of packet headers by combining multiple packets into one packet, consequently alleviating bandwidth requirements. The packet aggregation methods can be categorized into **timeout** and **quorum-based** types[31,9]. On one hand, timeout-based aggregation methods collect and propagate packets which were generated in a configured time period. On the other hand, quorum-based methods aggregate packets until a fixed number of packets are merged. This kind of aggregation assures the reduction of bandwidth requirements but does not limit packet transmission delay, contrary to the timeout-based approach.

3.3 Architectural and Middleware Support for MMOGs

Next we will characterize the different network supports used in MMOGs and how they can influence key aspects of the system.

3.3.1 Architectures

In this section we explore existing architectural solutions used in the gaming scenario, some commercially adopted and others academically used.

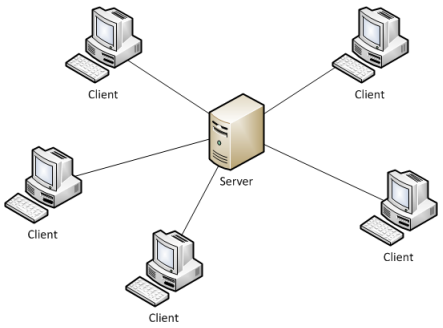

Fig. 6. Client-Server Architecture

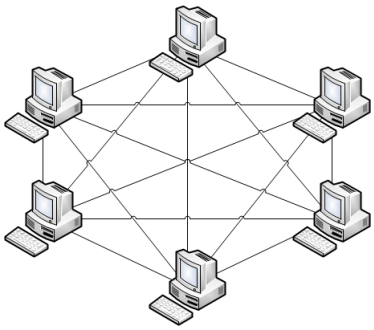

Fig. 7. Peer-to-Peer Architecture

Client-Server. In a CS architecture[26], every client communicates with a game server responsible for maintaining the current game state (Fig. 6). Each player sends its updates to the server which receives this information, resolves any possible consistency conflicts between concurrent updates and advances the game simulation, sending the resulting state back to the players.

This model became popular thanks to its simplicity of implementation, since there is only one connection per client, and state maintenance, since only the game engine in the server side generates game results and the results are globally identical among clients, making cheating a harder task. The CS model is especially advantageous to the game providers, who can easily control game subscriptions, spread game engine updates and monitor game activity.

There are some problems with the CS architecture however. First, the server's scalability is limited to its processing power and bandwidth capacity, thus it can only accept a small number of clients. Second, the server is a single point of failure, able to compromise the entire system if it becomes off-line. Moreover, clients may hold their

input procedure until the response from the server arrives and this can possibly prolong the network latency between the server and clients.

To increase the scalability of this solution and mitigate the bandwidth requirements many systems employ the distributed client-server model, also called the proxy-server model[19].

Peer-to-Peer. In the P2P gaming model[13,1], each player (peer) is responsible for executing its own game simulation, eliminating the need for a centralized server. To send updates to the system, a peer has to broadcast them to the system hoping to reach every other peer (Fig. 7).

The lack of a centralized server brings several advantages to the P2P architecture. First, it eliminates the single point of failure. This model replicates the game state through all peers, being able to overcome the disconnection of one or more peers. Second, it does not have a server bottleneck. Although the P2P architecture has a higher bandwidth requirement at the players than de CS architecture, that requirement increases linearly with the number of players. Also, there is no delay related with the exchange of updates with the server in the process of synchronization, reducing latency levels. Finally, the P2P model provides increasing scalability with the addition of new peers by assimilating their computational resources to the system.

However, the P2P model is not as popular as the CS in the gaming industry[26], mostly because they are complex of implement and configure due to the communication protocol between peers and the distributed agreement protocol needed to detect inconsistencies among updates. Many P2P solutions [12,13] use Pastry to map the participating nodes and the application objects, supporting object insertion and lookup, with a Scribe infrastructure to multicast messages. Another problem associated to the P2P model is the need to assign special responsibilities to peer nodes which can compromise the game play as each peer presents different computational and communication resources.

There are solutions, such as Peer-to-Peer with Central Arbiter[39], that merge the P2P and CS solutions (Fig. 8). In PP-CA, players exchange updates as they would in the P2P model. Moreover, each player sends his updates to a Central Arbiter. The role of the CA is the same as the server in the CS model but it only sends a correction update, instead of the latest game state to all clients when there are inconsistencies. If inconsistencies are rare events, the bandwidth requirements will be lower than in the CS model but higher than in the P2P model.

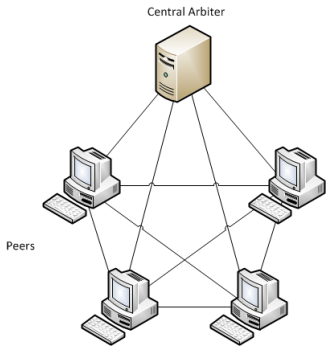

Fig. 8. Peer-to-Peer with Central Arbitrer Architecture

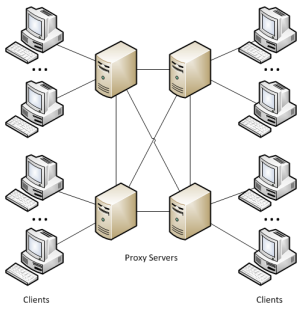

Fig. 9. Proxy-Server Architecture

Proxy-Server. The Proxy-Server model[19] or distributed Client-Server model was designed in an attempt to deal with the scalability restrictions of the centralized CS model by using several proxy servers. This model proposes that a client connects to the proxy server which is geographically closer to him and all servers are connected through a P2P structure.

When to submit an update, a client sends it to a proxy server, which sends back an acknowledgement, if the update is validated successfully, and forwards it to all other participant servers. Each proxy will then update his game state and, finally, when all proxies are updated, send the update to all his *avatars*.

Although this model offers greater scalability than the CS model and requires less bandwidth from clients, it presents a larger delay associated to the update process since it requires synchronization between proxies.

Independent from a specific topology, there are other architectural features that characterize a system, among them update propagation techniques. There are two major types of update propagation: **unicast** and **multicast**.

Unicast is a transmission technique that sends packets from one host to another at a time. Based on point to point communication, in unicast each process waits for messages through a port (or ports) and unique port numbers are assigned to the process for identification purposes. The most advantageous characteristics of unicasting are its simplicity of implementation and broad acceptance in network devices. However, compared to multicast, it requires more bandwidth to propagate packets to multiple nodes because each message, with the same content, has to be copied and transmitted to multiple destinies.

Multicast techniques[31], on the other hand, can transmit packets to multicast groups at the cost of sending one packet. To receive packets from the multicast group, the node has to subscribe (or join) it. The challenge in the design of a multicast-based application is how to categorize all transmitted information into multicast groups. Another problem related to multicast is that not every router allows this technique as they need to store the state of each multicast group.

In Table 2 we summarize the key aspects of each network topology.

	Topologies			
	CS	P2P	PP-CA	PS
Network Nodes	Server/Client	Peer	Peer/CA	Server/Client
Number of Servers	1	0	1	any
Number of Clients	limited	any	limited	any
Disposition	centralized	distributed	both	distributed
Point of Failure	single	multiple	both	multiple
Complexity	simple	complex	complex	complex
Consistency Authority	yes	no	yes	yes
Scalability	limited	yes	limited	yes
Update Dissemination	state	any	any/operation	any
Client CPU Load	low	high	high	low
Client BW Load	low	high	high	low
Client Communication	unicast	multicast	multicast	unicast
Server CPU Load	high	-	high	high
Server BW Load	high	-	very low	very high
Server Communication	multicast	-	unicast	multicast
Commercial Use	widely	little	no	no

Table 2: Network Architecture Comparison

3.3.2 Real-time Strategy Games

RTS games, first made popular by Dune 2[6] released in 1992, are generally characterized by three types of action: resource collection, unit construction, and battles between enemy units. In this class of games, the player assumes the role of a god like supreme commander, which commands different kinds of units presenting different capabilities through the map with the goal of defeating enemy troops and gaining control over the scenario. Today RTS games reached a huge popularity thanks to sagas like Warcraft[7], Command and Conquer[8] and Age of Empires[9], each one focusing on different realities: fantasy worlds, armed combat and human evolution. There are several design choices to attend when developing a strategy game, such as:

- ***real-time vs turn-based.*** One of the most decisive choices when developing a strategy game is to chose between real-time strategy and turn-based strategy. On one hand, turn-based strategy (TBS) ports the format of strategy boardgames, such as Risk[10], to the virtual environment. Each player can only make moves at his turn, having to wait for all remaining users to play before playing again, which can take a long time. Additionally, player's moves during his turn are limited. On the other hand, in real-time strategy (RTS), players can perform actions simultaneously and without limitations, bringing dynamism to the game. Although RTS is more realistic and appealing to most users, it needs consistency enforcing mechanisms not necessary in the TBS approach.

- ***map partitioning.*** Over the years, many map partitioning techniques have been employed. On Warcraft 2,[11] developers opted for partitioning the global map into square tiles, allowing objects to move to any adjacent square. TBS games, such as Civilization IV,[12] usually choose to partition the map using hexagon tiles. By partitioning the map, developers can easily calculate the distance between objects, ranges of attack and collisions between objects (when they occupy the same tile). However, this approach is not realistic. Nowadays, most RTS games do not partition the map, making object movements much smoother and realistic. Nevertheless, game engines have to detect collisions between objects.

- ***multiplayer partitioning.*** In order to avoid the scalability problems associated with centralized multiplayer management, many systems opt for distributed solution, partitioning the global map for all available servers. In the partitioning process, each server is assigned an area of the map with sizes depending on the network and communication resources available, guaranteeing the load balancing of the system. This way, servers with stronger resources are assigned with bigger areas and servers with weaker resources are assigned to smaller ones. Map partitioning is usually made into rectangular or hexagonal areas, however it disregards objects disposition. Other approaches try to partition the virtual world using computational geometry techniques, such as Voronoi Diagrams[36], that partitions the game space into polygonal regions based on locality properties. Although Especially effective in RTS game where object activity is frequently limited to a particular

[6] http://dune2k.com/duniverse/dune2
[7] http://www.blizzard.com/war3
[8] http://www.commandandconquer.com
[9] http://www.microsoft.com/games/empires/
[10] http://www.hasbro.com/risk/
[11] http://eu.blizzard.com/en-gb/games/legacy
[12] http://www.2kgames.com/civ4/

region, the time complexity associated with Voronoi Diagrams is a major disadvantage.

- ***2D vs 3D.*** Since the computer graphic evolution started, many legacy games, such as Warcraft, have developed titles that explored the 3D capabilities. While games like Warcraft 2 tried to explore a sense of 3D by picturing building and characters in perspective, the animation of objects through the map, done using sprite sheets, and limited user view of the scenario made clear it is a 2D game. The sequel, Warcraft 3, is already a full 3D game, where every entity is a 3D object and also the map. One of the greatest differences between Warcraft 2 and 3 is on the camera control. In Warcraft 3 the player have a wide spectrum of possible camera movements, providing great dynamic to the game.

  Next we establish a comparison between two popular strategy games: StarCraft and Civilization V, presenting the design choices of each one.

- ***StarCraft.*** Released in 1998, StartCraft,[13] is a space themed 2D RTS in which the results of the victory or defeat on the game will be decided by strategies of each player. The game map is characterized for being partitioned into square tiles. The main attraction of StarCraft is the network play. StarCraft presents its own dedicated game server system, Battle.Net, which works as a game lobby system that provides game sessions for players all around the world. Although players have to connect to Battle.Net to find a game session, it works only as an intermediate system, establishing a client-server connection between participants where one acts as the server. In essence it acts as a broker or a game server.

- ***Civilization V.*** Civilization V[14] is a 3D turn-based strategy game released in 2010. In this game, the player assumes the role of a grand commander and has to lead his nation for glory by conquering every other nation or surviving until the year 2050. As most TBS games, the game map is partitioned into hexagonal tiles and entities only exist in units, each with different combat strength depending on the number of battles it participated and vulnerabilities. Network playing is only possible using the STEAM[15] system dedicated server system.

3.4 Global Analysis

After exploring the state-of-the-art in the fields of consistency management, interest managing and architectural support, it is possible to conclude that consistency maintenance is achievable at low bandwidth costs by dynamically identifying the different consistency requirements objects present at runtime. This way, a model like VFC, which combines IM techniques with continuous consistency management, is a suitable choice for reducing bandwidth requirements and server CPU load by distributing it through the clients.

4 Solution

This section starts with the presentation of the proposed adaptation of the Vector-Field Consistency model to RTS game environment. Then, in section 4.2, we describe the

[13] http://us.blizzard.com/pt-br/games/sc
[14] http://www.civilization5.com
[15] http://store.steampowered.com

network architecture proposed followed by the software architecture to the *middleware* solution in section 4.3. Finally, we present the reasons that lead to the choice of *Warzone 2100* as the target game in section 4.4.

4.1 Adapting the VFC to RTS games

The proposed adaptation of the VFC model is composed of two parts: adaptation of the player's area-of-interest to the area of the map that is captured by his camera view and aggregation of entities into units according to the distance between them.

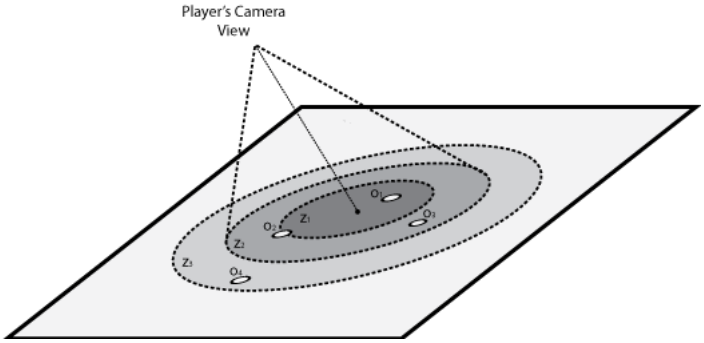

Fig. 10. Adaptation of player's AoI to his camera view

In Figure 10 we present an illustration of the proposed solution. By assigning the *pivot* point to the center of the player's camera view, we guarantee that every object within the player's area of visibility presents an acceptable consistency degree for the correct execution of the game. Furthermore, the flexibility of VFC and its *consistency zones* adds dynamism to the solution. By guaranteeing bounded consistency of the objects in the outskirts of the player's view, this solution makes it possible for the user to change the camera's position without compromising the user's experience. When this event takes place, the VFC's configurations take the new location into account and dynamically assure the consistency of the objects presented in the new AoI.

To further minimize the network traffic due to update propagation, we present the concept of entity aggregation into units. A unit is a group of entities controlled by the same player and presenting the same set of actions within a limited time period. Especially useful for position changing updates, if the system detects entities within a fixed distance threshold going on the same direction, it establishes a *pivot* point in the center of the unit formation and *consistency zones*. This process reduces the number of *pivot* points without compromising the consistency of the system, consequently reducing the processing and communication load.

4.2 Network Architecture

In this section we present a possible multiplayer partitioning solution (Fig. 11). In this distributed architecture, each server is assigned with a region of the map according to its network and processing capacities so that servers with more resources get bigger regions or with greater number of players and servers with short resources get smaller ones. Although the figure illustrates a distributed approach, a centralized architecture can easily be achieved by concentrating all the regions in one single server. Similarly, a

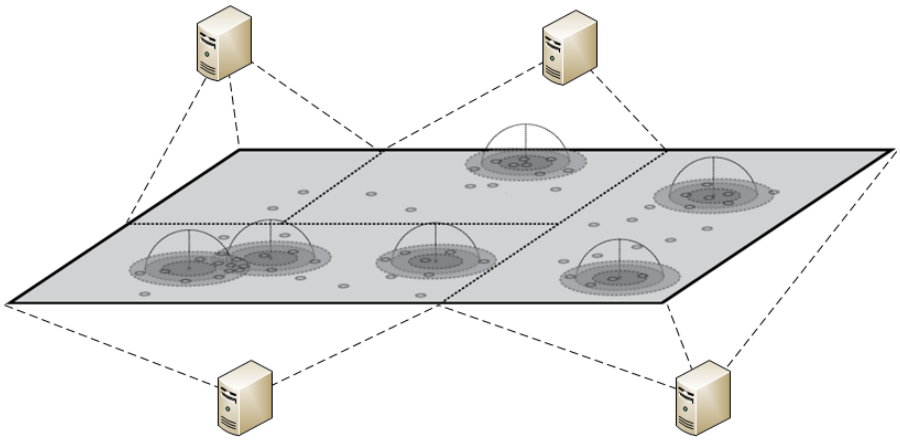

Fig. 11. Distributed network architecture

distributed approach may be clustered (e.g., with a coordinating server), which is our main focus, or fully decentralized following a peer-to-peer fashion.

4.3 Software Architecture

The software architecture can divided in two main components: the **server** component and the **client** component (Fig. 12). Each of this components is running the game engine which interfaces with the VFC *middleware* using an API. The server retains the primary objects and is responsible for maintaining their consistency while replicated objects are kept on the client side. The client-server communication is made by a Network Layer, making use of the Session Manager components to implement and assure the communication protocol.

The proposed solution operates as follows: Each client maintains a local replica of the collection of objects in the Object Pool, although with different consistency levels. Objects located within areas demarcated as areas-of-interest of the client present stronger consistency levels from objects outside the AoI. To know what objects require consistency maintenance, the VFC *middleware* communicates with the game, via API, to acquire game state changes and report them to the server. In the server, the global game state is updated with the client changes and the new game state is propagated to the remaining clients, according to their consistency requirements. Note that the client-server component approach referred in this paper does not strict the network topology choices for implementation. In a distributed architecture, P2P for instance, the server component would be allocated to a peer.

The solution considers two kinds of updates: **player objects** and **game objects**. Although the system does not differentiate this two types of objects, player objects, such as buildings and infantry, are the only ones that present strong continuous consistency, since they regard the player's immediate vision of the game. Additionally, changes to player objects are instantly locally processed and then send to the server, contributing to the dynamism of the system. Game objects, on the other hand, only need to be updated when they enter the player's AoI. This kind of objects comprehend environmental objects, such as mines or other resource facilities that change state during the game, and enemy objects. By making the distinction between objects, this solution contributes to reduction of the bandwidth load since most processing work is made in the client side.

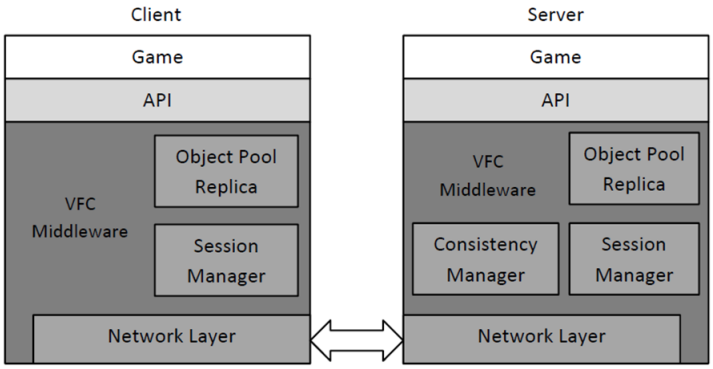

Fig. 12. VFC-RTS Software Architecture

The *middleware* (with software architecture depicted in Figure 12), will be implemented using C#, making use of the .Net interface for communication purposes. The reason for choosing C# was the fact that its easily compatible with the chosen game since it is written in C++ plus the vantages of the .Net library for communication. However, implementing the *middleware* in C# limits the spectrum of games that can use it to games implemented in C or C++.

Although the purpose of the work is to implement a *middleware* independent enough so that it can be applied to a large set of games, there are some points in the game code that need adaptation, specifically regarding communication and update exchange. In the code of *WarZone2100* is possible to identify some key files, such as: *"netqueue.cpp"* which is important for update exchange since it defines queues for incoming and outgoing messages; *"nettypes.h"* presents the struct definition for network messages and *"netsocket.cpp"* defines the establishment of client-server communication. In this work we intend to identify and explore the update exchange mechanism and, possibly, intercept them to be analyzed by the VFC-RTS *middleware*.

4.4 Game Choice

The reasons that led to chose *WarZone2100* as the target game for the implementation of the solution were:

- Entirely written in C++ and easily compiled.
- Wide library of maps with different dimensions.
- Comercial game originally developed by Pumpkin Studios and published by Eidos Interactive, now developed by the Warzone 2100 Project with open-source code available on [16].
- Still in development, with a large community and forum for developers.
- Full 3D environment, perfect to test the application of the VFC algorithm to a 3D environment.

5 Evaluation

The evaluation of the implemented solution can be divided in two classes: **qualitative evaluation** and **quantitative evaluation**.

[16] http://developer.wz2100.net

To perform a quantitative evaluation of the solution we intend to use the *GSM: Game Scalability Model*[19,20]. Designed to perform an analytical study about the scalability of network topologies in multiplayer real-time games, the GSM model provides the possibility to compare scalability levels of different network approaches and supplies detailed decision guidance based on the comparison results. Additionally, the GSM model can be incorporated into a particular game in order to self monitor the utilization of resources and estimate the maximum number of players at runtime.

To estimate the maximum number of players, GSM analyzes values such as:

- Time to receive, validate and process local client actions
- Time to update a single entity
- Time to send updates to a single client
- Time to send updates to other proxies (distributed solution)

However, additional criteria is important to contemplate when evaluating this particular system:

- Number of exchanged messages
- Savings in bandwidth and memory
- Selection of updates

The approaches we propose to implement and evaluate using GSM are:

- *WarZone 2100* using the native multiplayer system
- *WarZone 2100* with VFC-RTS applied to a centralized architecture
- *WarZone 2100* with VFC-RTS applied to a distributed architecture

To create a realistic test scenario for the quantitative evaluation we will use *BOTs* (artificial intelligent objects) in order to also estimate the maximum number of entities supported without affecting playability.

The qualitative evaluation of the solution will analyze the perception real players have of the game execution, focusing on aspects regarding playability. To perform the evaluation of results we intend to ask the participants to fill a questionary about their impressions of the game in each of the implemented approaches. Although we intend to perform the game simulations on a local network, internet conditions can also be simulated by introducing additional latency to the network.

6 Conclusion

In this paper we have presented VFC-RTS, a continuous consistency model based on the VFC model applied to the RTS class of games. The VFC model is characterized for exploiting the notion of locality awareness to establish spectrums of consistency.

First we made an overview about the state-of-the-art in the fields of consistency maintenance, interest management and RTS games, exploring and comparing the different approaches and existing solutions. Then we present our solution for the VFC-RTS system and propose an architecture for a generic *middleware*. A number of criteria were presented for a future analysis of an implementation of this work.

In the future, we will apply the VFC-RTS *middleware* to *WarZone2100*, a militaristic open-source RTS game to evaluate the solution.

References

1. D. Ahmed, S. Shirmohammadi, and J. de Oliveira. A hybrid p2p communications architecture for zonal mmogs. *Multimedia Tools and Applications*, 45:313–345, 2009. 10.1007/s11042-009-0311-y.
2. D. T. Ahmed and S. Shirmohammadi. A dynamic area of interest management and collaboration model for p2p mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.
3. J. a. Barreto, J. a. Garcia, L. Veiga, and P. Ferreira. Data-aware connectivity in mobile replicated systems. In *Proceedings of the Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '09, pages 9–16, New York, NY, USA, 2009. ACM.
4. C. E. Bezerra, F. R. Cecin, and C. F. R. Geyer. A3: A novel interest management algorithm for distributed simulations of mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 35–42, Washington, DC, USA, 2008. IEEE Computer Society.
5. J.-S. Boulanger, J. Kienzle, and C. Verbrugge. Comparing interest management algorithms for massively multiplayer games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA, 2006. ACM.
6. J. Carvalho, J. Oliveira, and P. Carvalho. Jogos on-line: Estudo sobre qoe e qos. In *CRC 2010 - 10a Conferência sobre Redes de Computadores*, pages 83–88, Universidade do Minho, Braga, Portugal, 2010. CRC 2010.
7. M. Claypool. The effect of latency on user performance in real-time strategy games. *Computer Networks*, 49(1):52 – 70, 2005. Networking Issue in Entertainment Computing.
8. J. F. F. d. Costa. Vector-field consistency for cooperative work. In *MSc Thesis 2010*, Instituto Superior Técnico, Lisboa, Portugal, 2010.
9. J. Dyck, C. Gutwin, T. C. N. Graham, and D. Pinelle. Beyond the lan: techniques from network games for improving groupware performance. In *Proceedings of the 2007 international ACM conference on Supporting group work*, GROUP '07, pages 291–300, New York, NY, USA, 2007. ACM.
10. C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *SIGMOD Rec.*, 18:399–407, June 1989.
11. T. A. Funkhouser. Ring: a client-server system for multi-user virtual environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pages 85–ff., New York, NY, USA, 1995. ACM.
12. T. Hampel, T. Bopp, and R. Hinn. A peer-to-peer architecture for massive multiplayer online games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA, 2006. ACM.
13. B. Knutsson, M. M. Games, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games, 2004.
14. N. Krishnakumar and A. J. Bernstein. Bounded ignorance: a technique for increasing concurrency in a replicated system. *ACM Trans. Database Syst.*, 19:586–625, December 1994.
15. L. Lamport. Time clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21:558–565, July 1978.
16. E. Lety, L. Gautier, and C. Diot. Mimaze, a 3d multi-player game on the internet. In *Proceedings of the 4th International Conference on Virtual System and MultiMedia*, 1998.
17. F. W. Li, L. W. Li, and R. W. Lau. Supporting continuous consistency in multiplayer online games. In *Proceedings of the 12th annual ACM international conference on Multimedia*, MULTIMEDIA '04, pages 388–391, New York, NY, USA, 2004. ACM.
18. B. F. Loureiro. Vfc-game. In *MSc Thesis 2010*, Instituto Superior Técnico, Lisboa, Portugal, 2010.
19. J. Müller and S. GORLATCH. Gsm: a game scalability model for multiplayer real-time games. *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, 00:2044–2055, 2005.
20. J. Müller and S. GORLATCH. Rokkatan: scaling an rts game design to the massively multiplayer realm. *Comput. Entertain.*, 4, July 2006.

21. J. Munson and P. Dewan. A concurrency control framework for collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, CSCW '96, pages 278–287, New York, NY, USA, 1996. ACM.
22. A. Negrão. Vfc large-scale: Consistency of replicated data in large scale networks. In *MSc Thesis 2009*, Instituto Superior Técnico, Lisboa, Portugal, 2009.
23. D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, UIST '95, pages 111–120, New York, NY, USA, 1995. ACM.
24. L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '02, pages 23–29, New York, NY, USA, 2002. ACM.
25. D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Softw. Eng.*, 9:240–247, May 1983.
26. J. D. Pellegrino and C. Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *Proceedings of the 2nd workshop on Network and system support for games*, NetGames '03, pages 52–59, New York, NY, USA, 2003. ACM.
27. Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37:42–81, March 2005.
28. N. Santos, L. Veiga, and P. Ferreira. Vector-field consistency for ad-hoc gaming. In R. Cerqueira and R. Campbell, editors, *Middleware 2007*, volume 4834 of *Lecture Notes in Computer Science*, pages 80–100. Springer Berlin / Heidelberg, 2007.
29. M. Shapiro and N. Preguia. Designing a commutative replicated data type. Technical report, Computer Science Dept: University of Copenhagen, 2007.
30. S. Singhal and M. Zyda. *Networked virtual environments: design and implementation.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.
31. J. Smed, T. Kaukoranta, and H. Hakonen. A review on networking and multiplayer computer games. In *In multiplayer computer games, proc. int. conf. on Application and development of computer games in the 21st century*, pages 1–5, 2002.
32. C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 59–68, New York, NY, USA, 1998. ACM.
33. C. Sun, X. Jia, Y. Zhang, and Y. Yang. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems. In *In Procs. of Intl. ACM SIGGROUP Conf. on Supporting Group Work*, pages 425–434. Press, 1997.
34. C. Sun, Y. Y., Z. Y., and C. D. A consistency model and supporting schemes for real-time cooperative editing systems. In *Proceedings of the 19th Australian Computer Science Conference*, pages 582–591, 1996.
35. D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 172–182, New York, NY, USA, 1995. ACM.
36. A. Tumbde. A voronoi partitioning approach to support massively multiplayer online games. Technical report, The University of Wisconsin, 2004.
37. L. Veiga, A. Negro, N. Santos, and P. Ferreira. Unifying divergence bounding and locality awareness inreplicated systems with vector-field consistency. *Journal of Internet Services and Applications*, 1:95–115, 2010. 10.1007/s13174-010-0011-x.
38. J. Vogel and M. Mauve. Consistency control for distributed interactive media. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 221–230, New York, NY, USA, 2001. ACM.
39. A. P. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, pages 99–104, New York, NY, USA, 2005. ACM.
40. H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20:239–282, August 2002.

7 Attachment - Work plan

	Month
Source code study	January
Design of communication API	February
Implementation of GSM model	March
Implementation of centralized VFC-RTS	March
Implementation of distributed VFC-RTS	April
Evaluation and debug	May - June
Performance analysis	June
Conclusion, review and delivery	July