A Framework for Inconsistency Analysis and Control of Massively Multiplayer Online Games

Zheng Da Wu
Bond University, Gold Coast, Australia
Email: zwu@bond.edu.au

***Abstract*— Inconsistency is a challenge with many delay-sensitive Massively Multiplayer Online Games (MMOGs). In this paper, a framework for inconsistency analysis and control is proposed. A typical situation, called hot-spot region, is considered for resolving this problem, since it is one of major causes of the inconsistency. Stochastic models are presented to abstract the operations of a MMOG in the context of our goal, particularly for examining the impact of various system characteristics and player behaviors on the inconsistency. A definition of game inconsistency together with a method for computing inconsistency rate is also provided. By using our models, we carry out inconsistency analysis quantitatively and obtain a deep insight into its behavior. Based on our analysis, we develop a mechanism for inconsistency control in the context of hot-spot region, so as to illustrate our approach in this area.**

***Index Terms*—massive multiplayer online games, federated MMOG p2p architecture, queuing theory and Markov processes, performance modeling.**

I. INTRODUCTION

IT is recognized that one of the key issues with Massively Multiplayer Online Games (MMOGs) is the delay for transmitting and processing an event or state-update triggered by a player during game-play [1][2][3]. For example, the first person shooter (FPS) games are thought to be the most delay-sensitive networked games. The games are fast paced while the players widely move around shooting and ducking. It would be extremely frustrating for the players who try to hit a target where there is an apparent delay between pressing triggers and weapons firing. The delay also causes another problem, called game inconsistency. A typical example of the inconsistency is "a dead man who is able to shoot" problem [4][5][6]. It is recognized that the inconsistency becomes one of key challenges for many MMOGs design. Therefore, the work to be described in this paper is aimed at the analysis and control for this problem. During the last five years, significant research has been reported in the context of this issue. The work dealt with the design of efficient communication architectures [7]-[12], interest management group [2][13][14], game-world partitioning [16][17][18], dead reckoning [19][20], message aggregation and compression [2][3][13]. The distinction of our work from the previous publications is argued as the followings:

1) A targeted condition: During a game session, players work on their missions or quests. In order to achieve their goals, they travel around the virtual world (or game world) and interact with each other. Their behaviors are random not only in temporal dimension but aslo in spatial dimension. It is observed that the popularity of different parts, called regions or rooms usually, of the virtual world is not identical for many games. For example, based on the measurement of well-known Quake-II's workload online, it is noticed that players tended to move between popular "waypoint" regions in the map and the popularity distribution of waypoint was "Zipf-like" [12][21]. About 30% of players gathered in only 1% of popular place. For example, the players congregate inside one of the buildings. This special place is often called *hop-spot region.* In such condition, the population of players is relatively large and interaction among them is intensive. Thus, the system workload for communication and processing is suddenly increased when a hot-spot region occurs. Consequently, unexpected delay and inconsistency occur. This paper will particularly deals with this issue.

2) Methodology: Most of existing works in the area of MMOGs deal with the design and implementation of a small size prototype with fixed network configuration based on a specific game so as to evaluate the proposed ideas or deal with the performance evaluation based on measurements from online game systems. It is relatively rare in the study of performance modeling and analysis for MMOGs with arbitrary configuration in terms of various system parameters. In this paper, stochastic models are originally presented so as to abstract the characteristics of a MMOG, particularly for investigating the impact of various factors on the system inconsistency. A formal definition of system inconsistency is proposed together with the mathematical expression derived for calculating the inconsistency rate. By carrying out performance analysis using our models, we can obtain a deeper insight into the behaviors of the inconsistency, which is further used in the design of our inconsistency control mechanism.

Manuscript received March 26, 2009; revised August 12, 2009; accepted September 10, 2009.

doi:10.4304/jcm.4.10.803-814

3) Control system design: Our strategy for the inconsistency control is characterized in three aspects: (i) the control design is aimed at a hot-spot condition, as discussed before; (ii) the control is dynamically performed in a real-time manner using online measures; (iii) the control is realized by performing co-operation between clients and servers.

The rest of this paper is organized as follows: In Section 2 we provide the description of a federated P2P MMOG architecture. In Section 3, we present the details of modeling and analysis of the system, including numerical examples. In Section 4 a mechanism for inconsistency control is proposed in the context of hot-spot conditions. The related works are reviewed in Section 5. Finally, concluding marks are disussed.

II. System Descriptions

A typical *game world* of MMOGs is made up of landscape (or terrain), player characters (or avatars), objects, and non-player characters (NPCs). The landscape consists of all immutable components in the game. As computer graphics they are generated by the client software pre-installed. A player experiences a game world through a game *avatar*, which represents his/her character, such as knight, wizard, and prince/princess in *Lineage* [24]. The state of a player includes the player's position in the virtual world and other attributes, such as possession, health or strength, intelligence or wisdom. Mutable *objects* are weapon, food and various tools. The NPCs, such as monsters, residents and teleporters, are computerized characters and controlled by their algorithms. They can be allies or enemies. During a game session, a player may take successive actions [9]. For examples, the player can control the avatar to move around the virtual world for visiting different rooms or fields, pick up weapons or armors, kill monsters, and fight with other players. These actions lead game state to change.

We choose a federated peer-to-peer architecture as a target for our study, since it is a typical and potential architecture proposed recently and attracting more attention [8][9][22][25]. It is a hybrid solution of client-server and peer-to-peer (P2P) technologies as shown in Fig. 1. Its game world is partitioned into *regions* based on the limited sensing capabilities of a player's avatar. Each region is associated with an *interest management group*, which is composed of all the players within this region [8][25]. By using a peer-to-peer (P2P) communication mechanism, the players (or peers) disseminate their state updates relevant to that region. When a player moves from one region to another, the player's interest group is changed. The system consists of three kinds of nodes: a central server, region servers, and peers. The *central server* (or *world server*) is responsible for partitioning a game world into regions and keeps the mapping between regions and interest groups. It helps peers (clients or players) to discover their groups and to receive the game data whenever they log-in the game or perform transition from one region to another. Thus, the function of the central server is independent of the nature of the game as most of game logic is executed at the clients.

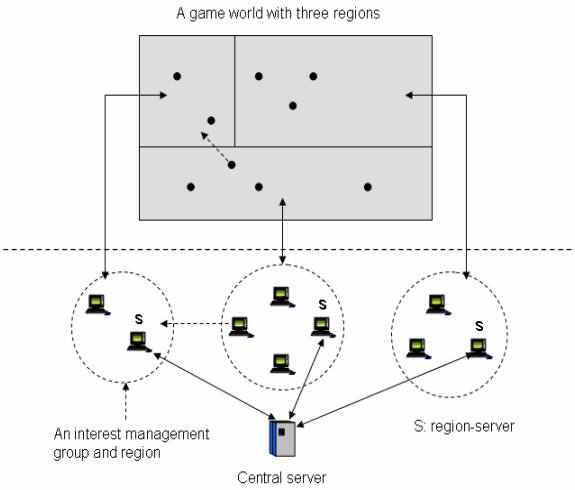

Figure 1. Federated peer-to-peer architecture

All-game communications are carried out in a P2P fashion once a player is connected to his interest group. *Region server* is a manager of an interest group associated with one region. It coordinates all shared objects or data related to that region. For example, it plays a role of arbiter for fighting events generated by the peers. As a root of the group broadcast tree, this server is responsible for disseminating data of region map and state-updates. A node of the broadcast tree, except for the root, is a *peer* in the region group. A game is a large state machine [3][25]. The game state must be consistent among the players of a region. In order to keep state-updates synchronously among the peers in a region, the peers send their relevant state changes to the region server, called *events* in this paper. The region server correlates them and then broadcasts state-updates to all the peers in that region. Once the peers receive the state-updates, each of them will generate a new scene of the game accordingly.

III. Modeling and Analysis

A. Models

Although the MMOGs support a large number of concurrent players, a game session is actually performed based on interest management groups or regions at a time. The characteristics of timing with each region are most essential for the whole game system design, since all events handling and state-updates disseminating are *region-oriented* in a federated P2P MMOG. There is no interaction between any two players, who are located in different regions at a time. As a result, we consider one region and its context of the game world as our targeted system for modeling. This does not loss any generality, since the implementation of each region follows the same protocol [22][25]. We abstract the targeted system as an $M^{[X]}/G/1$ queuing system [26], as illustrated in Fig. 2.

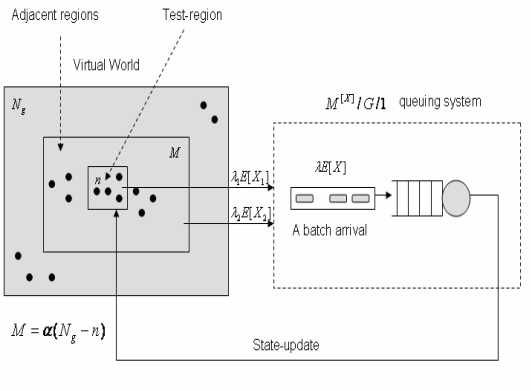

Figure 2. Batch arrivals and queuing model

In this queuing model customers, representing messages (or events) generated from clients of the region and the central sever, arrive in batches in accordance with a time-homogenous Poisson process with parameter λ. The batch size X is a random variable, $1 \le X \le N_g$, where N_g is the maximum number of players with a game. The assumption of the batch arrival is based on the fact that a game is a finite-state-machine and the game-state is updated periodically in most game implementation. During each period, the region server may receive multiple messages from the players or from the central server. Assume that the service takes place singly and that the service time for each customer is a random variable, denoted by υ , having a general distribution with mean $1/\mu$. In this model, the traffic intensity is $\rho = \lambda E[X]/\mu$, where $\rho < 1$. Thus, the steady-state is reached. Consider a test-customer regardless of where it comes from and let D be the sojourn time of the customer in the queuing system. The delay D, seen by the test-customer, consists of two independent delays, D_1 and D_2. D_1 is the delay or waiting time of the first member to be served of the batch in which the rest arrives, and D_2 is caused by the service time of the members of this batch that are served prior to the test-customer. Therefore, $D = D_1 + D_2$. The details for finding the distribution of sojourn time D and its expectation $E[D]$ in an $M^{[X]}/G/1$ queuing system are given in the Appendix. Accordingly, we can obtain

$$E[D] = \frac{1}{2(1-\rho)}\left\{\frac{E[X^2]-E[X]}{\mu^2} + E[X]E[\upsilon^2]\right\}\left\{\frac{E[X^2]}{E[X]} - 1\right\}\frac{1}{2\mu} \quad (1)$$

Since there are two kinds of arrivals in batches, the batch size X is the sum of two sub-sizes, X_1 and X_2, where $1 \le X_1 \le n$ and $1 \le X_2 \le M = \alpha(N_g - n)$. Specifically, n is the number of players who are in the test-region during a game-period. M is the number of players in the neighbor-regions of the test-region, and some of them potentially transit into the test-region as indicated in Fig. 2. We introduce factor α, $0 \le \alpha \le 1$, to reflect a probability of these "foreign players" who potentially invade the test-region. In fact, X_1 represents the number of messages generated by the "local players" in the test-region due to their taking actions. X_2 is the number of messages sent from the central server. This is because a message is generated by the central server whenever a foreign player transits into the test-region during a game period [22][25].

We assume the sub-size X_1 is binomially distributed and its probability mass function is

$$b(X_1; n; p) = \binom{n}{X_1} p^{X_1}(1-p)^{n-X_1} \quad (2)$$

Using this distribution, we can adjust the traffic workload from the clients to the region server by varying the values of two parameters, n and p, where p , $0 < p < 1$, represents the probability of each player sending a message to the region server. In other words, p represents the probability of a player taking an action (movement or interaction). As a result, the expectation and variance of X_1 are given respectively by

$E[X_1] = np$, $Var[X_1] = np(1-p)$ and thus,

$$E[X_1^2] = Var[X_1] + E[X_1]^2 = np[(n-1)p + 1] \quad (3)$$

Since the rate, at which a foreign player transits into the test-region, is not independent of the number of players in the test-region for most of games. We can model this transition process as a Markov Modulated Poisson Process (MMPP) [29], wherein the state of Markov chain covering the MMPP in one-to-one correspondence with the number of customers present in the system; that is if there are i customers (or players) in the system (i.e., region) at time t, the Markov chain governing the MMPP is in state i, and the arrival rate (or the transition rate) to the system is λ^i, $i \in \{0,1,\cdots,M\}$. Suppose that the residence times of a player in one region are exponentially distributed with mean γ. We denote by $\mathbf{Q}$ the infinitesimal generator for the Markov chain governing the arrival process. Then, $\mathbf{Q}$ has the following form:

$$\mathbf{Q} = \begin{bmatrix} -\lambda^0 & \lambda^0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\lambda^1 & \lambda^1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -\lambda^{M-2} & \lambda^{M-2} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -\lambda^{M-1} & \lambda^{M-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \quad (4)$$

where $\lambda^i = \lambda_2(M-i)$, $0 \le i \le M$, and $\lambda_2 = 1/\gamma$ represents the average transition rate. Once $\mathbf{Q}$ is specified, we can determine the *steady-state* distribution of the Markov chain, which is denoted by a vector $\boldsymbol{\pi} = (\pi_0, \pi_1, ..., \pi_M)$,

where π_i represents the stationary probability when the Markov chain have their equilibrium solution by solving the following equations [26]:

$$\begin{cases} \boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0} \\ \boldsymbol{\pi} \cdot \mathbf{e} = 1 \end{cases} \quad (5)$$

where $\mathbf{e} = [1,1,...,1]$ is a column vector with all its elements equal to unity. Consequently, the distribution of random variable X_2 is equivalent to the distribution of the Markov chain's steady-state. Thus,

$$E[X_2] = \sum_{i=0}^{M} i \cdot \pi_i \quad (6)$$

$$E[X_2^2] = \sum_{i=0}^{M} i^2 \cdot \pi_i \quad (7)$$

Since X_1 is generated by internal players of the region and X_2 is produced by the central server due to the foreign players' mobility, we may assume that X_1 and X_2 are independent with each other. Thus, we can obtain

$$E[X] = E[X_1] + E[X_2] \quad (8)$$

$$E[X^2] = E[X_1^2] + E[X_2^2] + 2E[X_1]E[X_2] \quad (9)$$

According to additive property of Poisson processes [26], we can have

$$\lambda E[X] = \lambda_1 E[X_1] + \lambda_2 E[X_2] \quad (10)$$

as shown in Fig. 2. Using equations (1)-(10), we can now evaluate the average sojourn time of a customer in the queuing system $E[D]$, once ρ, γ, μ, N_g, n, p and α are specified. Using Little's Law [28], we can also obtain the expected number of customers in the system by

$$E[N] = \lambda E[D] \quad (11)$$

B. Inconsistency

We denote system latency by T. It is identified that the latency T includes the following delays: d_1 - the elapsed time between an event-message being emitted from a peer and received by the region server; d_2 - the delay incurred by the event-message waiting and processing in the region server; It is noted that a state-update is generated by the server at the end of this time interval; d_3 - the elapsed time between the update-state message being issued by the region server and received by the peer; d_4 - the latency over which the peer processes the update-state and displays a new scene on the screen. Thus, we obtain $T = \sum_{i=1}^{4} d_i$. In this paper, we define the concept of inconsistency in terms of two metrics as follows:

1). T_h - the time interval between two successive actions taken by a player. Since there may be a sequence of actions triggered by other players during interval T_h, we can express it by $T_h = k\lambda^{-1}$, where k represents the number of actions happened during interval T_h and λ^{-1} is the average inter-arrival time of the event-messages. λ is also shown in Fig. 2. By this definition, we can interpret T_h as *thinking-time* of a player, which is proportional to k, once λ is given.

2). T - the system latency as analyzed above. Recall (11) and apply Little's law, we can get the expected number of customers (or messages) departed from the region server during period T and T_h by $N_T = \lambda T$ and $N_{Th} = \lambda T_h$, respectively. If $N_T \le N_{Th}$, it means that the player is able to see what happens on his screen in time whenever he triggers an event. In this case, there is no scene or display lost in the game progress, since the game state-update caused by his action can be completed before he takes next action. Otherwise, in the case of $N_T > N_{Th}$, the player will miss ($N_T - N_{Th}$) state-updates triggered by other players and thus, his decision for next action is actually based on uncompleted state-update. As a consequence, the player may experience a wrong outcome during game play. We call such scenario as *consistency loss* or *inconsistency*. Depending on the degree of user tolerance and game logic design, there may be different way to evaluate consistency loss rate. In this paper we estimate the inconsistency rate C_{loss} by

$$C_{loss} = \begin{cases} \dfrac{N_T - N_{Th}}{N_T}, & if \quad N_T > N_{Th} \\ 0, & otherwise \end{cases} \quad (18)$$

Since $N_T = \lambda T$, $N_{Th} = \lambda T_h$ and $T_h = k\lambda^{-1}$, the inconsistency rate can be simply expressed by

$$C_{loss} = \begin{cases} 1 - \dfrac{k}{\lambda \cdot T} & if \quad k < \lambda \cdot T \\ 0 & otherwise \end{cases} \quad (19)$$

C. Analysis

It is recognized that the system delay is one of key causes for consistency loss during game-play [2][4][6]. This is quantitatively reflected in (19). Recall equation $T = \sum_{i=1}^{4} d_i$, we can express it as $T = T_1 + T_2$, where $T_1 = E[D]$ represents the average queuing delay, including service time at region server, and T_2 the rest of network delay. As discussed before, the delay T_1 depends on the service rate $1/\mu$ and the traffic workload $\lambda E[X]$, which is further determined by the number of players within the region, the number of players around the region, their mobility behavior, and the intensity of their interaction, which have been reflected in our modeling. In contrast with T_1, T_2 is caused by the network and local computer a player chooses for a game play. It is noted that such facilities are independent of the game world design and player behaviors.

In what follows, we present numerical examples to evaluate the system delay and inconsistency rate in terms

of the game system characteristics based on our models above. It means that we ignore the impact of T_2, since it is independent of the game design and player's behavior. We assume that the distribution of the service time in $M^{[X]}/G/1$ queuing system is exponential with mean $1/\mu$. Accordingly, under the same other assumptions as given in Section 3.*A*, we can first compute the average sojourn time of a customer in the system, based on (1), by

$$E[D] = \frac{\rho}{\lambda(1-\rho)}\left\{\frac{E[X^2]+E[X]}{2E[X]}\right\} \qquad (20)$$

Then, we evaluate inconsistency rate C_{loss} using (19), where $T = E[D]$. We employ the queuing model to investigate inconsistency in the context of game-design and players-behavior, excluding the facilities players choose. Fig. 3 shows the impact of traffic intensity ρ on the system delay T with different value of n, the number of players in the region, where we specify $N_g = 36$, $\alpha = 0.1$, $p = 0.4$, $1/\mu = 0.001\,s$, $\gamma = 10\,s$ and $k = 4$. Fig. 4 demonstrates the feature of inconsistency rate C_{loss} under the same assumptions as given in Fig. 3. From Fig. 4, we identify that (i) when ρ is small, say $\rho \le 0.6$, no inconsistency occurs; (ii) the inconsistency rate is increasing when the value of n is getting large under the same traffic intensity. Fig. 5 presents the performance of T via ρ with varying the value of α, where we fix $N_g = 36$, $n = 4$, $p = 0.4$, $1/\mu = 0.001\,s$, $\gamma = 10\,s$ and $k = 4$. Fig. 6 illustrates the behavior of inconsistency under the same assumptions as given in Fig. 5. From Fig. 6 we note that when α, the probability of foreign players moving to the region, is increased, C_{loss} is also increased significantly.

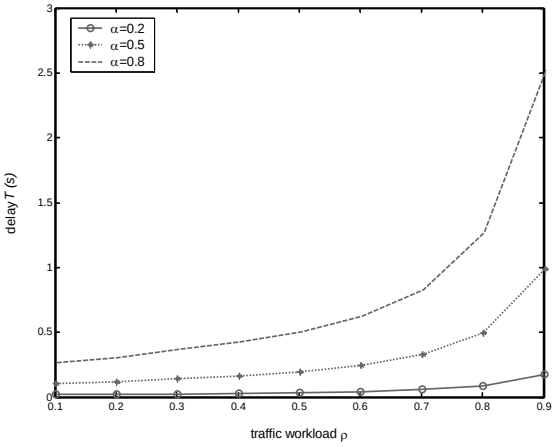

Figure 5. Delay analysis with varying α against ρ

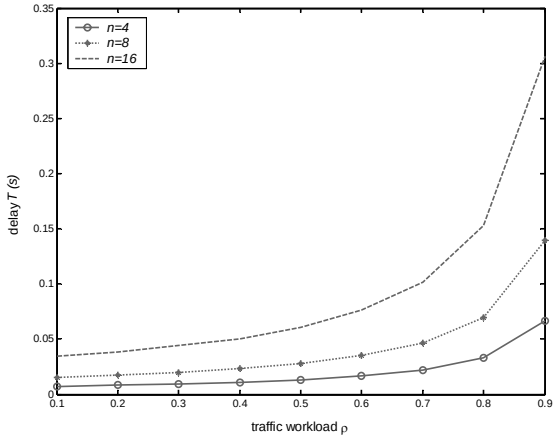

Figure 3. Delay analysis with varying n against ρ

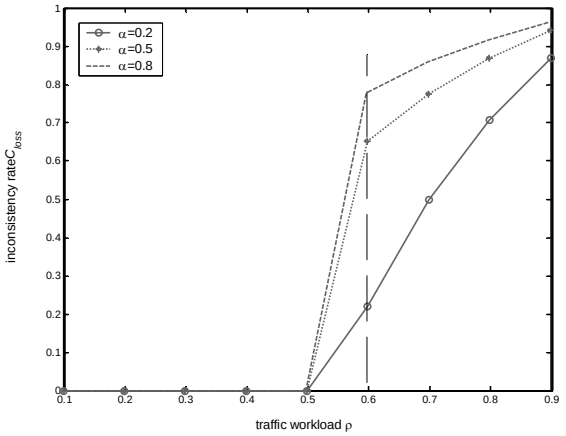

Figure 6. Inconsistency analysis with varying α against ρ

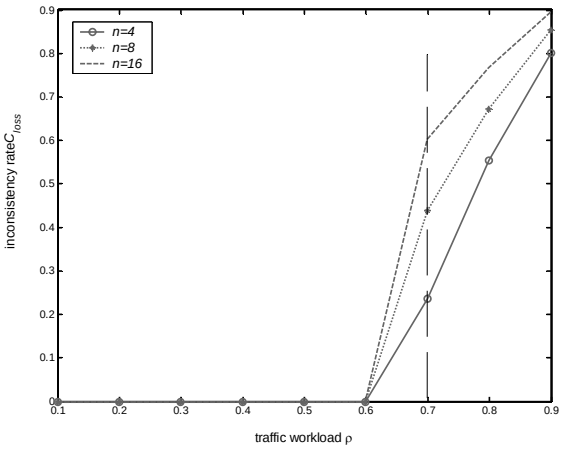

Figure 4. Inconsistency analysis with varying n against ρ

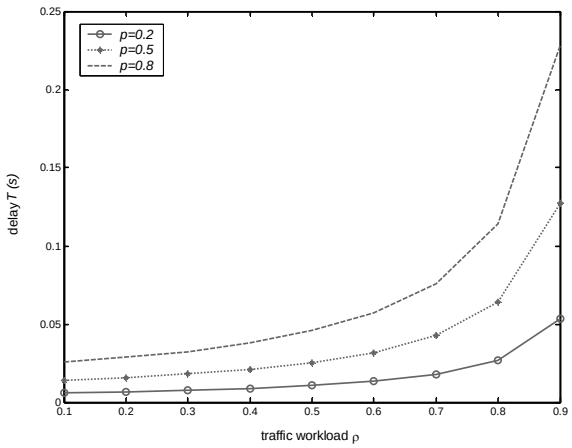

Figure 7. Delay analysis with varying p against ρ

Fig. 7 demonstrates the impact of p on the system delay T, where $N_g = 36$, $n = 6$, $\alpha = 0.1$, $1/\mu = 0.001\,s$, $\gamma = 10\,s$ and $k = 4$. Recall that p is the probability of a player triggers an event caused by his movement or interaction.

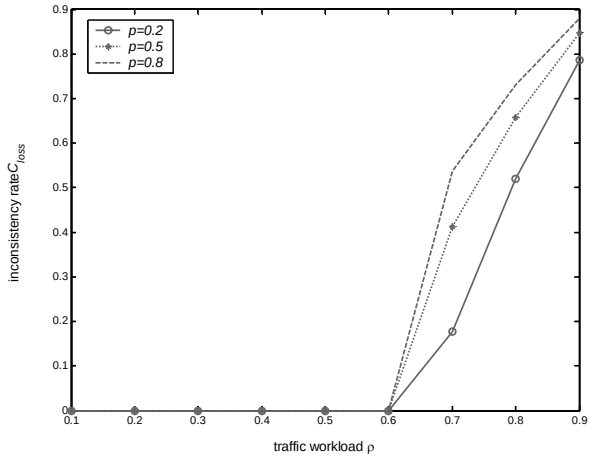

Figure 8. Inconsistency analysis with varying p against

Fig. 8 illustrates the feature of consistency loss under the same assumption as given in Fig. 7. We find that C_{loss} is increased with increment of p. This means that the activity of a player, such as his movement or interaction, is one of important factors to affect the value of C_{loss}.

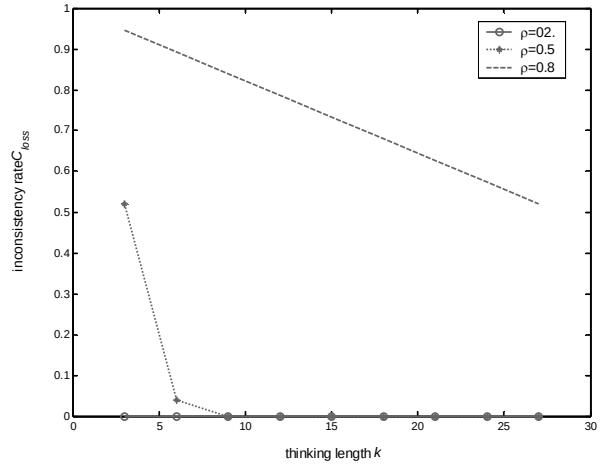

Figure 9. Inconsistency analysis with varying ρ against k

Finally, we investigate the relationship between C_{loss} and k, the length of thinking-time, under different traffic intensity. It is shown in Fig. 9, where we set $N_g = 36$, $n = 10$, $1/\mu = 0.001\,s$, $\gamma = 10\,s$, and $\alpha = p = 0.4$. We recognize that (i) the inconsistency rate is decreased with increasing the thinking-time; (ii) the inconsistency rate is high when the traffic intensity is heavy.

IV. INCONSISTENCY CONTROL

A. Control System Design

In order to perform inconsistency control, we first need a mechanism for detecting an inconsistency risk timely during game-play. Recall the definition of consistency-loss and (18) for calculating the inconsistency rate C_{loss}, we find $C_{loss} > 0$ only if $N_T - N_{Th} > 0$ or $T - T_h > 0$, since $N_T = T\lambda$ and $N_{Th} = T_h\lambda$. We assume that each client c_i can take measure for T_i and T_i^h during game-play. The measures, as part of data in the packets together with original information generated by client c_i, are sent to the region server whenever the player takes an action. Similar to the definitions of T and T_h, T_i is the response-time experienced by client c_i and T_i^h represents "thinking-time" of client c_i. As a result, we can define $\delta_1 = T_i - T_i^h$ as a *risk indicator*. If $\delta_1 > 0$, it indicates that an inconsistency occurs. Since T_i^h is determined by player's behavior, we cannot control it from the system designer's point of view. Recall the system delay $T = T_1 + T_2$ presented in section 3.*C*, where T_1 represents the average queuing delay and T_2 is the network delay. This leads us to consider inconsistency control in different ways for the two cases. In this paper, we only concentrate on the inconsistency control due to queuing delay, especially for a hot-spot region. In such situation, the population of players is relatively large and interaction among them is intensive. Therefore, it leads to a heavy workload for the region server. Consequently, unexpected delay and inconsistency occurs.

We propose a mechanism, called *local-division handling*, which can approximately reduce the queuing delay to a half in the context of a hot-spot region. We define δ_2 as another indicator to show when the local-division handling is applied during game-play. Specifically, we define

$$\delta_2 = \begin{cases} 1, & if \quad T_i - \frac{1}{2}Q\lambda^{-1} \le T_i^h \\ 0, & otherwise \end{cases} \qquad (21)$$

where λ and Q are the average arrival rate and the average queuing length for a certain period respectively, which can be estimated by online measures. They will be addressed in detail soon. The control is realized by performing co-operation between clients and region servers. The control mechanism at each region server is illustrated in Fig. 10.

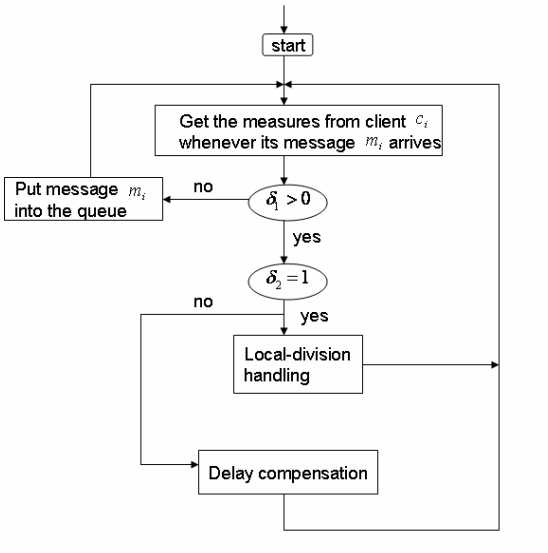

Figure 10. Framework flow of the control system

Since a hot-spot region occurs randomly and it is of temporary nature, we apply the control only if $\delta_2 = 1$, i.e., $T_i - \frac{1}{2}Q\lambda^{-1} \le T_i^h$. It means that in such situation the queuing delay T_1 is dominant in the system delay T, comparing to the network delay T_2. It is obvious that only reducing queuing delay cannot resolve the inconsistency problem completely, especially when the queuing delays do not play a key role in the cause of unexpected system delay. In other words, the unexpected delay may be mainly caused by heterogeneous environment. Therefore, we need to apply for some *delay compensation* technique, which is beyond our scope in this paper. We are currently working on this solution.

B. Online Measures

In order to handle the messages from clients and coordinate these data at region server, each region server maintains a data structure or table which contains the state-information for its players [8]. We extend the data structure to including some online measures associated with inconsistency control. The extended online measures are illustrated in Table 1.

TABLE I.
EXTENDED ONLINE MEASURES

Clients, c_i	Distance, h_i	Total frequency of events, f_i	Thinking-time, T_i^h	Response-time, T_i
c_1	h_1	f_1	T_1^h	T_1
c_2	h_2	f_2	T_2^k	T_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_n	h_n	f_n	T_n^h	T_n

In this table, we still use client IDs, c_i, $i = 1,2,...,n$, as indices for searching its data. h_i represents the round-trip-delay between client c_i and the region server. The value of h_i can be measured whenever player (or client) c_i joins the region. By finding $h^* = \min\{h_i\}$, $i = 1,2,...,n$, the region server can always find a client c^*, $c^* \in \{c_1, c_2, \cdots, c_n\}$, which has the minimum delay h^* to it. Client c^* will be employed as an assistant region server when a hot-spot condition occurs, which will be discussed in the next subsection. f_i represents the frequency of messages generated by client c_i to the region server, which can be determined by recording the number of messages, m_i, issued by client c_i during a period τ. For a game design, a popular measure of how fast an animation progress is *frames per second* (FPS). The desired 100 FPS imply that each iteration of the game animation loop should take 10ms. Thus, we may set $\tau = a(FPS)^{-1}$, where a is an integer, say $a = 10$. Accordingly, $f_i = m_i / \tau$, which indicates the average arrival rate of the messages from c_i for period τ. In addition, the region server estimates the total arrival rate of messages λ and the average queue length Q for each period τ by calculating $\lambda \approx \sum_{i=1}^{n} f_i$ and $Q \approx [\sum_{i=1}^{\lambda\tau} q_i / (\lambda\tau)] + 1$ respectively, where q_i is the number of waiting messages in the queue when message i is received by the server.

C. Control Algorithms

Given the online measures in Table 1, the region server can execute the algorithm for local-division handling, as indicated in Fig. 10, which is further described as below.

Control Algorithm at Region Server

1. Whenever $\delta_2 = 1$ occurs
2. Save the current value of λ and Q as $\tilde{\lambda}$ and $\tilde{Q}$ respectively
3. Put message m_i into the queue for processing
4. Call *local division procedure*, as indicated in Fig.10
5. Two region servers S_1 and S_2 start for managing the region in parallel
6. **For** S_1 (or S_2)
7. Initialize $i = 0$
8. check status flag δ_2 immediately after a message is processed
9. **If** $\delta_2 = 0$
10. **Then** $i = i + 1$
11. **Repeat** line 8-10 **Until** $i = 10$
12. Call *local mergence procedure*

The local division procedure called in the algorithm

above is further illustrated in Fig. 11, where a hot-spot region is originally managed by server S_1 as shown in the case of (a). When an inconsistency risk

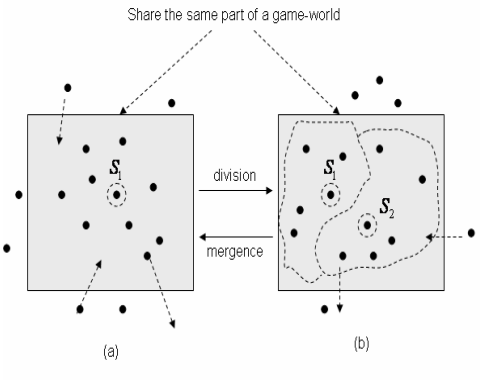

Figure 11. Region division and mergence

occurs (due to $\delta_1 > 0$) and the queuing delay needs to be reduced to a half (due to $\delta_2 = 1$), the region is divided into "two virtual regions" as indicated in the of case (b), where an assistant server S_2 is appointed. It is noted that the two regions actually share the same part of the game world based on its original partitioning scheme. In other words, the two virtual regions play the same role in the region management. The only difference is that each region server has only a half of the number of players from the original region and they work collaboratively so as to improve the efficiency of messages processing and delivery during game-play. The specific procedure for the division is presented as the follows:

- Server S_1 selects one of its clients in the region as an additional server S_2 such that $S_2 = \{c_i \mid h_i = h^*, i = 1,2,...,n\}$, where h^* was defined as before. This rule makes communication between the two servers with the shortest delay.
- Sort the list of current frequencies f_i in Table 1 in descendant order and obtain $\tilde{f}_1 \geq \tilde{f}_2 \geq \cdots \geq \tilde{f}_n$. Divide the clients into two groups using the sorted list, denoted by G_1 and G_2, such that $G_1 = \{c_i \mid c_i = \tilde{f}_i, i = 1,3,5,...,n \text{ or } n-1\}$, where index i must be an odd numbers and $G_2 = \{c_i \mid c_i = \tilde{f}_i, i = 2,4,6,...,n \text{ or } n-1\}$, where index i must be an even number. This separation is fair for the two new regions in terms of the number of players and workload for each region.
- S_1 sends the current state-information and Table 1 to S_2, including $\tilde{\lambda}$ and $\tilde{Q}$, which will be used for local mergence procedure late.
- S_1 notifies the address of S_2 to the central server and all the clients in G_2. Then, S_2 starts working with its members as a new group in a normal way.
- Each time when S_1 (or S_2) generates a state-update, it sends the update to S_2 (or S_1) at first and then broadcasts its members. Once S_2 (or S_1) receives an update, it coordinates its state information for keeping all the states consistent in the two servers, since they still share the same part of the game-world.
- Whenever a foreign player moves in this hot-spot region, the central server notifies the new member to S_1 and S_2 alternatively so as to achieve fair workload for each server.

The algorithm can support the local division procedure recursively in principle. However, it will rarely happen in practice, as the risk indicator can notify the inconsistency risk timely.

Since a hot-spot region is of temporary nature, the two virtual regions may be merged late. This is because that (i) some players are killed; (ii) some players move away from the current region; (iii) the intensity of interaction between players is relieved due to (i) and (ii). Therefore, a procedure for *local mergence* is included in the algorithm. In order to distinguish the online measures for the two servers, we now denote the messages arrival rate λ as $\lambda_{S1}(\text{or}\,\lambda_{S2})$ and the queue length Q as $Q_{S1}(\text{or}\ Q_{S2})$ for server $S_1(\text{or}\ S_2)$ respectively. Also, we define an indicator β, $\beta \in \{0,1\}$ to show when the local mergence procedure is invoked during game-play. β can be evaluated by two flags, β_1 and β_2, which are expressed by

$$\beta_1 = \begin{cases} 1, & if \quad \lambda_{S1} + \lambda_{S2} \leq \frac{1}{2}\tilde{\lambda} \\ 0, & otherwise \end{cases} \text{ and}$$

$$\beta_2 = \begin{cases} 1, & if \quad Q_{S1} + Q_{S2} \leq \frac{1}{2}\tilde{Q} \\ 0, & otherwise \end{cases}$$

where, $\tilde{\lambda}$ and $\tilde{Q}$ represent the message arrival rate and the queue length for a hot-spot region server just before it is divided, respectively. If $\beta = \beta_1 \cap \beta_2 = 1$, the mergence procedure will be triggered. This condition is quite conservative. Also, the operations prior to calling the mergence procedure are proposed in line 7-11 of the control algorithm at each region server. Therefore, we can ensure that there will be no possibility for the hot-spot region to take place again in a short time.

Local Mergence Procedure

1. **For** server S_1 (or S_2)
2. Send the current measures $\lambda_{S1}(\text{or}\,\lambda_{S2})$ and $Q_{S1}(\text{or}\,Q_{S2})$ to $S_2(\text{or}\ S_1)$ carried by state-update messages
3. Server S_2 (or S_1) receives the measures and

calculates β

4. **If** $\beta = 1$, **Then** server S_2 (or S_1) transfers its leadership to server S_1 (or S_2)

It is noted that the leadership transfer can be achieved in the same way as one region server transits to other region [8]. Therefore, no additional protocol is needed here.

This control algorithm significantly reduces the inconsistency rate by using parallel computation and communication with two region servers for a hot-spot region. It can be verified using our numerical results in the last section. As indicated in Fig. 5, when $\rho = 0.7,$ we find that $C_{loss}(n = 4) \approx 0.24$, $C_{loss}(n = 8) \approx 0.44$, and $C_{loss}(n = 16) \approx 0.61$. In Fig.7, when $\rho = 0.6,$ we recognize that $C_{loss}(\alpha = 0.2) \approx 0.22$, $C_{loss}(\alpha = 0.5) \approx 0.65$, and $C_{loss}(\alpha = 0.8) \approx 0.78$. In addition, the enhanced mechanism only operates when an inconsistency risk occurs without affecting the original design and implementation of a game. However, the algorithm produces overhead in data processing and communication. This is worth for users to enjoy their game-play by avoiding inconsistency occurrence.

V. RELATED WORK

John C.S Lui and M. F. Chan [16] propose an efficient partitioning algorithm for Distributed Virtual Environments (DVEs). In this paper the authors discussed major challenges in designing a scalable, cost-effective, and high performance DVE system, formulated a number of functions for the avatar's workload to the system computational cost, in order to carry out the system performance evaluation and obtain their optimal partitioning algorithms. Daniel Bauer et al. [25] propose a very general model for evaluating the scalability of massive multi-player games with three communication architectures: client-server, peer-to-peer and federated peer-to-peer. They define the cost of operations on input processing resource and network processing resource. This is the first paper that gives a quantitative assessment of these architectures by using analytical models. E. Rhalibi and M. Merabti [22] propose a fully distributed, peer-to-peer architecture for MMOGs. The authors specify the architecture by using agent-based modeling technique. However, quantitative performance evaluation is not given in the paper. In the paper [18] by Li Zou et al, the authors propose their static and dynamic simulation models for the performance evaluation of game state dissemination with cell-based and entity-based strategies. In the paper some basic concepts for distributed games, such as entity, player, cell, vision domain, player area and interaction group, are well discussed. M. Ye and L. Cheng [20] present a method for modeling MMORPGs (massively multiplayer online role-playing games) system performance and applied it in the analysis of two real MMORPGs. The results show that a strong linear relationship exists between performance metrics at server side and the number of currently players online. The performance model can be used for resource allocation at runtime. E. Lety et al [17] propose an approach at the transport layer, using multiple multicast groups and multiple agents, to achieve dynamic partition optimally for large-scale virtual environment. They use a method based on the theory of planner point processes for performance analysis of their system design.

One of key differences from the existing works as mentioned before is that in our paper the MMPP and $M^{[X]}/G/1$ stochastic models are used for the analysis of system delay and inconsistency, rather than using the cost of system resources consumed as performance metrics.

S. Rooney et al [8] present a federated P2P network game architecture, including detailed discussion for the design of multicast reflectors, one kind of region servers, and transport protocols, in order to improve the system performance in terms of synchronization, delay and loss. B. Knutsson et al [9] present an excellent paper in the topic of P2P support for MMOGs. The design and implementation of their prototype, *SimMud* game, is introduced. In addition, the basic concepts for MMOGs together with the issues for P2P infrastructure games are well discussed in this paper. A. Bharambe et al [12] present the design, implementation and evaluation of Colyseus, a distributed architecture for interactive multiplayer games, which takes advantages of a game tolerance, for weakly consistent state and predictable workload to meet the tight latency constraint of game-play and maintain scalable communication cost.

We benefit the works from the last three papers for studying inconsistency issue in the context of their communication architectures. The work presented in our paper is to enhance the existing architectures for solving the inconsistency problem caused by the two conditions without modifying their original design and implementation.

VI. CONCLUSIONS

We have proposed a framework for inconsistency analysis and control in the context of a hot-spot region. Compared to the existing works in this area, our new contributions are: (i) we originally presented analytical models to abstract the operations of a federated P2P MMOG so as to perform the system delay and inconsistency analysis quantitatively; (ii) we proposed a definition for the concept of inconsistency together with an expression for computing inconsistency rate in terms of the system characteristics and players activities; (iii) based on our models and the numerical results , we proved the fact that the longer delay a game system experiences, the higher risk consistency-loss occurs. Especially, we identified that three factors, the number of players in a region n, the mobility pattern factor α, and the activity of players in the region p, directly impact on the system delay and consistency-loss; (iv) our strategy for the design of inconsistency control is based on dynamically monitoring the inconsistency status by using our derived formulas with online measures. The strategy keeps the original design and implementation of a game

and thus, the control algorithms are only operated as enhanced mechanisms.

As analyzed in this paper, unexpected delay may be caused by heterogeneous environment. This problem has to be resolved by *delay compensation* technique. We are currently working on this solution as our future outcome in this area.

APPENDIX: THE $M^{[X]}/G/1$ MODEL WITH BULK ARRIVAL

This appendix presents the details for finding the distribution of sojourn time D in an $M^{[X]}/G/1$ queuing system and determining its expectation $E[D]$.

A. The number in the system at departure epochs in steady state (Pollaczek- Khinchin formula)

Assume that the arrival epochs occur in accordance with a Poisson process with rate λ and the number of arrivals at each epoch is given by a random variable (RV) X having distribution $a_j = \Pr(X = j)$, and the probability generation function (PGF)

$$A(s) = \sum_j a_j s^j \quad \text{and} \quad E(X) = \sum j a_j = A'(1) = a \tag{1}$$

$\{a_j\}$ is the batch size distribution.

The total arrivals A constitute a compound Poisson process having PGF $\exp\{-\lambda[1-A(s)]\}$. Suppose that N is the total number of arrivals during the service time of a customer. Then the PGF of N is given by

$$E[s^N] = K(s) = B^*[\lambda - \lambda A(s)] \tag{2}$$

where it is assumed that the service times are independent and identically-distributed RV having a general distribution with the probability density function (PPDF) $B(t)$ and mean $(1/\mu)$. Thus, $B^*(s) = \int_0^\infty e^{-st} dB(t)$ is its Laplace-Stieltjes-Transform (LST), then $-B^{*(1)}(0) = 1/\mu$.

The traffic intensity is $\rho = \lambda E(X)/\mu = \lambda a/\mu$. Assume that $\rho < 1$ so that the steady state is reached. Since we can have the Pollaczek- Khinchin (P-K) for $M/G/1$ as the follows:

$$V(s) = \frac{(1-\rho)(1-s)B^*(\lambda - \lambda s)}{B^*(\lambda - \lambda s) - s} \tag{3}$$

where $V(s)$ is the PGF of the number of customers leaves behind in the system when a customer is departing. The (P-K) formula can now be extended for $M^{[X]}/G/1$ by

$$V(s) = \frac{(1-\rho)(1-s)B^*(\lambda - \lambda A(s))}{B^*(\lambda - \lambda A(s)) - s} \tag{4}$$

In other words, $V(s)$ is the PGF of the number of customers in the $M^{[X]}/G/1$ system at departure epochs in steady state. In particular case when $a_1 = 1$, $a_j = 0$, $j > 1$, we get $A(s) = s$ and $K(s) = B^*(\lambda - \lambda s)$, then we have an $M/G/1$ queue.

B. Waiting –time distribution

Burke [27] obtained the waiting-time distribution in an $M^{[X]}/G/1$ queuing system in 1975. What follows is based on his approach.

Consider a test unit and let D be the total waiting time of the unit in queue. Namely, D is the queuing time of an arbitrary test unit. The delay D is seen by the test unit to consist of two independent delays, D_1 and D_2. D_1 is the delay (or waiting time) of the first member to be served of the batch in which the test unit arrives, and D_2 is the delay caused by the service times of the members of this batch that are served prior to the test unit. In other words, $D = D_1 + D_2$. Let W and W_i be the PDF of D and D_i, $i = 1,2$, respectively, and let $W^*(s)$ and $W_i^*(s)$ be the LST of W and W_i, respectively. Let $B(t)$ be the service-time distribution and $B^*(t)$ = LST of the PDF of the total service time of all customers belonging to the same arrival group. Then

$$\beta^*(s) = \sum_{k=1}^{\infty} a_k [B^*(s)]^k = A[B^*(s)] \tag{5}$$

To find the delay D_1, consider a batch as a whole as a single super customer. Then the LST of the waiting time of the first customer of the batch in which the test unit arrives can be obtained from the corresponding expression of an $M/G/1$ system with $B^*(s)$ replaced by $\beta^*(s)$. That is, if $\rho = \lambda a/\mu < 1$, then replacing $B^*(s)$ by $\beta^*(s)$, we get

$$W_1^* = \text{LST of the delay } D_1 = \frac{s(1-\rho)}{s - \lambda[1 - A(B^*(s))]} \tag{6}$$

Let p_i be the probability that the test customer arrives in a batch of size of *i*. Let *K* be significantly large number. Then in the first *K* batches of arrivals, the number of batches with i arrivals will be approximately $a_i K$, $i = 1,2,\cdots$ and the total number of customers arriving in batch of size i will be approximately $ia_i K$. Thus, the total number of arrivals in *K* batches is

$$\sum_{i=1}^{\infty} i a_i K$$

and the proportion of those arriving in batches of size *i* is

$$\frac{ia_i K}{\sum_i i a_i K} = \frac{ia_i}{\sum_i i a_i} = \frac{ia}{a}$$

where $a = E(X)$. Thus, for large *K*,

$$p_i = \frac{ia_i}{a} \ . \quad (7)$$

Assume now that the test customer arrives in a batch of size i. Assume further that service within members of any batch is in random order. Then the probability that the test customer chosen in the *j*th in the batch of i is $1/i$, $j = 1,2,\cdots, i$. Again, if he/she is the *j*th customer to be taken for service, his/her delay(or waiting time in the queue) will be equal to the service time of $(j-1)$ customers of the batch (of size i) in which he/she arrives and who are served prior to him/her. Now conditioning on the size of the batch *i* on which the test customer arrives, we get

$$P(D_2 \le t) = W_2(t) = \sum_{i=1}^{\infty} \Pr\{delay \le t \mid he\ arrives\ in\ a\ batch\ of\ size\ i\}\, p_i$$

$$= \sum_{i=1}^{\infty}\left[\sum_{j=1}^{i} B^{(j-1)*}(t)\frac{1}{t}\right]p_i \quad (8)$$

where B^{k*} is the *k*-fold convolution of *B* with itself. Thus,

$$\begin{aligned} W_2^*(s) &= \text{LST of } W_2(t) \\ &= \sum_{i=1}^{\infty}\frac{p_i}{i}\{\sum_{j=1}^{i} \text{ LST of } B^{(j-1)*}(t)\} \\ &= \sum_{i=1}^{\infty}\frac{ia_i}{ia}\{\sum_{j=1}^{i}[B^*(s)]^{j-1}\} \\ &= \sum_{i=1}^{\infty}\frac{a_i}{a}\frac{1-[B^*(s)]^i}{1-B^*(s)} \\ &= \frac{1-A[B^*(t)]}{a[1-B^*(s)]}\ , \end{aligned} \quad (9)$$

Since $D = D_1 + D_2$, the LST of the total delay *D* or waiting time in the queue of the test customer that has the LST given by

$$\begin{aligned} W^*(s) &= W_1^*(s)W_2^*(s) \\ &= \frac{s(1-\rho)}{s-\lambda+\lambda A[B^*(s)]}\frac{1-A[B^*(s)]}{a[1-B^*(s)]}\ . \end{aligned} \quad (10)$$

The waiting time in the system or response time of the test unit is given by $W_s = D + \upsilon$, where υ is the service time. Thus, the LST $W_s^*(s)$ of W_s, the response time, is given by

$$W_s^*(s) = W^*(s)B^*(s)\ .$$

C. Moments of $D = D_1 + D_2$ for $M^{[X]}/G/1$

We can have

$$E(D) = E(D_1) + E(D_2) \text{ and}$$

$$E(D_1) = -\frac{d}{ds}W_1^*(s)|_{s=0} = -\frac{d}{ds}\frac{s(1-\rho)}{s-\lambda+\lambda A[B^*(s)]}\Big|_{s=0}$$

$$= -(1-\rho)\frac{s-\lambda+\lambda A(B^*(s))-s\{1+\lambda A'(B^*(s))\frac{d}{ds}B^*(s)\}}{\{s-\lambda+\lambda A[B^*(s)]\}^2}\Big|_{s=0}$$

(which is of the form $0/0$). Using L'Hopital's rule and simplifying, we get

$$E(D_1) = \frac{\lambda(1-\rho)}{2(1-\rho)^2}\frac{d^2}{ds^2}A[B^*(s)]\,|_{s=0}$$

where $(d^2/ds^2)A[B^*(s)]$ is the second moment of the supercustomer's service time. On simplification, we get

$$E(D_1) = \frac{\lambda}{2(1-\rho)}\{A''[B^*(s)]\}\left[\frac{d}{ds}B^*(s)\right]^2 + A'[B^*(s)]\frac{d^2}{ds^2}B^*(s)\Big|_{s=0}$$

$$= \frac{\lambda}{2(1-\rho)}\left[A''(1)\left(-\frac{1}{\mu}\right)^2 + A'(1)\mu_2{}^2\right]$$

where $\mu_i = E(\upsilon^i)$. Writing $a^{(2)} = E(X^2)$, we get $A''(1) = E(X^2) - E(X) = a^{(2)} - a.$
Thus,

$$E(D_1) = \frac{\lambda}{2(1-\rho)}\left[\frac{a^{(2)}-a}{\mu^2} + a\mu_2\right]. \quad (11)$$

Again,

$$E(D_2) = -\frac{d}{ds}W_2^*\big|_{s=0} \text{ or}$$

$$aE(D_2) = -\frac{d}{ds}\frac{1-A[B^*(s)]}{[1-B^*(s)]}\Big|_{s=0}$$

$$= \frac{\{A[B^*(s)]\frac{d}{ds}B^*(s)\}[1-B^*(s)]-\{1-A[B^*(s)]\}\frac{d}{ds}B^*(s)}{[1-B^*(s)]^2}\Big|_{s=0}$$

(which is of the form $0/0$). Using L'Hopital's rule and simplifying, we get

$$E(D_2) = \frac{1}{2\mu}\left(\frac{a^{(2)}}{a} - 1\right) \quad (12)$$

Thus,

$$E(D) = \frac{1}{2(1-\rho)}\left(\frac{a^{(2)}-a}{\mu^2} + a\mu_2\right) + \left(\frac{a^{(2)}}{a} - 1\right)\frac{1}{2\mu} \quad (13)$$

It is noted that equation (13) above is directly used for determining *E[D]* with $M^{[X]}/G/1$ model, which is equation (1) in our submitted paper [26][28].

ACKNOWLEDGMENT

The work was supported by the Australian DEST research funds and the Vice-Chancellor's Research Scheme 2008 of Bond University.

REFERENCES

[1] Tristan Henderson, The effects of relative delays in networked games, *PhD thesis*, Department of Computer Science, University College London, February 2003.

[2] G. Armitage, M. Claypool, and P. Branch, Networking and Online Game –Understanding and Engineering Multiplayer Internet Games , (John Wiley & Sons, 2006).

[3] A. Davison, *Killer Game Programming in Java*, (O'Reilly, 2005).

[4] Martin Mauve, How to keep a dead man from shooting, Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, pp.199-204, October 17-20, 2000.

[5] J. D. Pellegrino & C. Dovrolis, Bandwidth requirement and state consistensy in three multiplayer game architectures, *ACM Proc. of NetGames*, Redwood City, CA, 2003, 52-59.

[6] W. Palant, C. Griwodz, and P. Halvorsen, Consistency requirements in Multiplayer Online games, *Proceedings of ACM Netgames'06*, October 30-31, 2006.

[7] D. Bauer, S. Rooney and P. Scotton, Network infrastructre for massively distributed games, *Proc. of NetGames 2002 Conference*, Braunschwig, Germany, 2002, 3-9.

[8] S. Rooney, D. Bauer & Rudy Deydier, A Federation peer-to-peer network game architecture, *Research Report, IBM Zurich Research Laboratory, RZ 3528* (#99542), 2004.

[9] B. Knutsson, H. Lu, W. Xu & B. Hopkins, Peer-to-peer support for massively multiplayer games, *Proc. of IEEE INFOCOM*, Hong Kong, 2004, 96-107.

[10] S. Fiedler, M. Wallner & M. Weber, A Communication architecture for massive multiplayer games, *ACM Proc. of NetGames*, Braunschweig, 2002, 14-22.

[11] M. Castro, P. Druschel, A. Kermarrec & A. Rowstron, A large-scale and decentralized application-level multicast infrastructure, *IEEE Journal on Selected Areas in communications (JSAC), 20*(8), 2002.

[12] A. Bharambe, J. Pang, and S. Seshan, Colyseus: A Distributed Architecture for Online Multiplayer Games, *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation*, pp. 155-168, 2006.

[13] J. Smed, T. Kaukoranta & H. Hakonen, A review on networking and multiplayer computer games, *Technical Report No 454*, (Turku Centre for Computer Science, April 2002).

[14] J. Boulander, J. Kienzle, and C. Verbrugge, Comparing Interest Management Algorithms for Massively Multiplayer games, , *Proceedings of ACM Netgames'06*, October 30-31, 2006.

[15] T. Limura, H. Hazeyama, Y. Kadobayashi, Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games, *ACM SGCOMM'04 Workshops*, Portland, Oregon, USA, 2004.

[16] J. C.S. Lui & M. F. Chan, An efficient partitioning algorithm for distributed virtual environment systems, *IEEE Trans. On Parrallel and Distributed Systems*, *13*(1), 2002, 1-19.

[17] E. Lety, T. Turletti, and E. baccelli, SCORE: A Scalable Communication Protocol for Large-Scale Virtual Environments, *IEEE/ACM Transactions on Networking*, April 2004 Volume 12, No. 2.

[18] L. Zou, M. H. Ammar & C. Diot, An evaluation of grouping techniques for state dissemination in networked multi-user games, *Proc. of 9th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication System (MASCOTS)*, 1999, 33-40.

[19] S. Aggarwal, H. Banavar, S. Mukherjee, and S. Rangarajan, Fairness in Dead-Reckoning based Distributed Muli-Player games, *Proceedings of the 4th ACM network and System Support for Games*, Hawthorne, NY, USA, October 2005.

[20] Y. Zhang, L. Chen, and G. Chen, Globally Synchronized Dead-Reckoning with Local lag for Continuous Distributed Multiplayer games, *Proceedings of ACM Netgames'06*, October 30-31, 2006.

[21] F. C. Wu, F. Chang, W. Feng & J. Wapole, A traffic characterization of popular online games, *IEEE/ACM transactions on Networking, 13*(3), 2005, 488-500.

[22] E. Rhalibi & M. Merabti, Agent-based modeling for a peer-to-peer MMOG architecture, *ACM Computer in Entertainment*, *3*(2), Article 3B, 2005.

[23] H. Fujinoki, On the Support for heterogeneity in Networked Virtual Environment, *Proceedings of ACM Netgames'06*, October 30-31, 2006.

[24] LINEAGE © Ncsoft. http://www.lineage.com/nic

[25] D. Bauer, I. Iliadis, S. Rooney & P. Scotton, Communication architectures for massive multiplayer games, *J. Multimedia Tools and Applications, 23*, Kluwer Academic Publishers, 2004, 47-66.

[26] J. Medhi, *Stochastic models in queuing theory*, (Second Edition, Academic Press, 2003).

[27] P.G. Burke, Delays in single server queues with batch input, *Operations Research*, **14**, 33-40.

[28] L. Kleinrok, Queuing systems, volume 1, theory, (Wiley, 1975).

[29] J. N. Daigle and M. N. Magalhaes, Analysis of Packet Networks Having Contention-based Reservation with Application to GPRS, IEEE/ACM Transactions on Networking, August 2003 Volume 11, No. 4, pp. 602-615.

[30] Z.D. Wu, Performance modelling of multicast groups for multiplayer games in peer-to-peer networks, *Proc. of 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, Montreal, Canada, 2005, 105-122.

[31] C. E. Palazzi, S. Ferretti, S. Cacciaguerra & M. Roccetti, Interactivity-Loss Avoidance in Event Delivery Synchronization for Mirrored Game Architectures, IEEE Transactions on Multimedia, Vol. 8, No. 4, August 2006, 874-879.

Zheng Da Wu received the Master degree of computer technology in the Graduate School of University of Science and Technology of China, Beijing, from the Chinese Academy of Sciences in 1981, and the PhD degree in computer science from University of Kent at Canterbury, U.K., in 1987. He is currently an associate professor of computer science in the School of Information Technology, Bond University, Gold Coast, Australia. His current research is in the area of mobile computing, multimedia communications and massive multiplayer networked games. He is a member of the IEEE.