

Simulazione di un ambiente virtuale distribuito




Fabio Biselli

Università di Bologna
Laurea Magistrale in Informatica
Corso di Simulazione di Sistemi

Bologna, 27 Luglio 2015

- 1 Introduzione
- 2 Caratterizzazione e Setup
- 3 Implementazione del modello
- 4 Simulazione ed analisi dei risultati
- 5 Conclusioni

Studi preliminari

-  M. Zyda, *From Visual Simulation to Virtual Reality to Games*, IEE Computer Society, September 2005;
-  S.A. van Houten, P.H.M. Jacobs, *An Architecture for Distributed Simulation Games*, Proceedings of the 2004 Winter Simulation Conference;
-  C. Ghosh, R.P. Wiegand, B. Goldiez, T.Dere, *An Architecture Supporting Large Scale MMOGs*, Proceedings of the 3rd International ICST Conference on Simulation Tools and Technique, 2010.

Gli articoli in sintesi

- Creare una scienza dei giochi ed una “Research Agenda”;
- La tecnologia è considerata la “spinta” che guida lo sviluppo e l’implementazione di un’architettura per un DVE;
- I requisiti per l’architettura le 3 U: **usefulness**, **usability** e **usage**;
- Proposta di un’architettura di giochi distribuiti;
- Proposta di un’architettura di giochi distribuiti su larga scala basata sul concetto di “overlapping zone”.

Improving the Performance of DVE Systems



P.Morillo, J.M.Orduna, M.Fernandez, and J.Duato.

Improving the Performance of Distributed Virtual Environment Systems.

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 16(7), 2005.

Improving the Performance of DVE Systems

- *Data Model*: describe alcuni metodi per distribuire dati persistenti o semipersistenti in un DVE;

Improving the Performance of DVE Systems

- *Data Model*: describe alcuni metodi per distribuire dati persistenti o semipersistenti in un DVE;
- *Communication Model*: analizza i metodi con cui gli avatar comunicano tra di loro;

Improving the Performance of DVE Systems

- *Data Model*: descrive alcuni metodi per distribuire dati persistenti o semipersistenti in un DVE;
- *Communication Model*: analizza i metodi con cui gli avatar comunicano tra di loro;
- *View Consistency*: mira ad assicurare che ogni avatar che condividono un'area comune abbia la medesima percezione degli oggetti presenti;

Improving the Performance of DVE Systems

- *Data Model*: describe alcuni metodi per distribuire dati persistenti o semipersistenti in un DVE;
- *Communication Model*: analizza i metodi con cui gli avatar comunicano tra di loro;
- *View Consistency*: mira ad assicurare che ogni avatar che condividono un'area comune abbia la medesima percezione degli oggetti presenti;
- *Network Traffic Reduction*: mantenere un basso numero di messaggi permette ai sistemi DVE di scalare in modo efficiente con il numero degli avatar connessi.

Risultati di P.Morillo et al.

Intenti

Valutazione del **Partitioning Problem** che è un punto chiave per il design di DVE scalabili. L'obiettivo è individuare un modo efficiente per assegnare a più server la gestione degli avatar.

Risultati di P.Morillo et al.

Intenti

Valutazione del **Partitioning Problem** che è un punto chiave per il design di DVE scalabili. L'obiettivo è individuare un modo efficiente per assegnare a più server la gestione degli avatar.

Risultati

- Assenza di correlazione ed un comportamento non-lineare in relazione al numero degli avatar della **funzione di qualità** proposta in letteratura;
- con nuovo metodo di partizione, basato sul bilanciamento del carico dei server, è possibile mantenere il carico al di sotto della soglia in cui le prestazioni medie del DVE degradano velocemente.

Obiettivi del progetto

- P.Morillo et al. propongono un modello di DVE producendo alcuni risultati importanti;
- grazie a questi è si propone un nuovo modello in cui si trascura il problema del carico dei server;
- si focalizza l'attenzione sul traffico di rete e la comunicazione;
- c'è una relazione tra numero di utenti e le prestazioni?

Il modello di P.Morillo et al.

- 3 server, di cui uno contrassegnato come principale;

Il modello di P.Morillo et al.

- 3 server, di cui uno contrassegnato come principale;
- 180 client che controllano un avatar nel mondo virtuale;

Il modello di P.Morillo et al.

- 3 server, di cui uno contrassegnato come principale;
- 180 client che controllano un avatar nel mondo virtuale;
- una rete che connette i client ai server (che sono tra loro interconnessi);

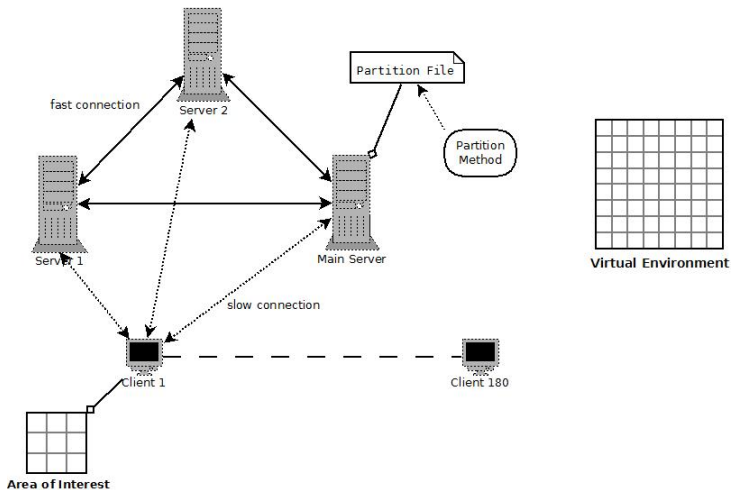
Il modello di P.Morillo et al.

- 3 server, di cui uno contrassegnato come principale;
- 180 client che controllano un avatar nel mondo virtuale;
- una rete che connette i client ai server (che sono tra loro interconnessi);
- un ambiente virtuale (Virtual Environment);

Il modello di P.Morillo et al.

- 3 server, di cui uno contrassegnato come principale;
- 180 client che controllano un avatar nel mondo virtuale;
- una rete che connette i client ai server (che sono tra loro interconnessi);
- un ambiente virtuale (Virtual Environment);
- un metodo ed un file di partizionamento che il server principale utilizza per suddividere il carico tra i server.

Il modello di P.Morillo et al.



Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;

Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;
- $k \in \{1, \dots, 9\}$ Server di Partizione che si occupano di gestire i messaggi tra client ed aggiornare il Main Server;

Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;
- $k \in \{1, \dots, 9\}$ Server di Partizione che si occupano di gestire i messaggi tra client ed aggiornare il Main Server;
- n client che controllano un distinto avatar nel mondo virtuale;

Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;
- $k \in \{1, \dots, 9\}$ Server di Partizione che si occupano di gestire i messaggi tra client ed aggiornare il Main Server;
- n client che controllano un distinto avatar nel mondo virtuale;
- una rete WAN simulata che connette i client ai server;

Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;
- $k \in \{1, \dots, 9\}$ Server di Partizione che si occupano di gestire i messaggi tra client ed aggiornare il Main Server;
- n client che controllano un distinto avatar nel mondo virtuale;
- una rete WAN simulata che connette i client ai server;
- una rete LAN simulata che connette i server con una struttura ad anello unidirezionale;

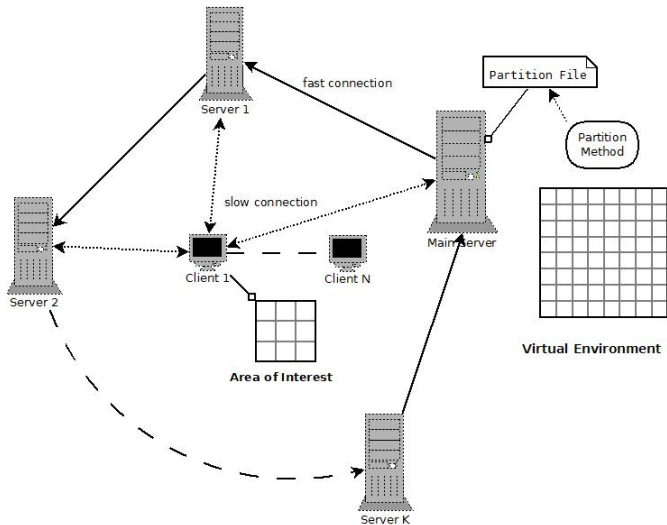
Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;
- $k \in \{1, \dots, 9\}$ Server di Partizione che si occupano di gestire i messaggi tra client ed aggiornare il Main Server;
- n client che controllano un distinto avatar nel mondo virtuale;
- una rete WAN simulata che connette i client ai server;
- una rete LAN simulata che connette i server con una struttura ad anello unidirezionale;
- un ambiente virtuale (VirtualEnvironment);

Il modello di progetto

- 1 Main Server che si occupa del login dei client e della gestione dell'ambiente simulato;
- $k \in \{1, \dots, 9\}$ Server di Partizione che si occupano di gestire i messaggi tra client ed aggiornare il Main Server;
- n client che controllano un distinto avatar nel mondo virtuale;
- una rete WAN simulata che connette i client ai server;
- una rete LAN simulata che connette i server con una struttura ad anello unidirezionale;
- un ambiente virtuale (VirtualEnvironment);
- un metodo di partizionamento statico, dipendente dal numero di Server di Partizione, che il Main Server utilizza per suddividere il carico.

Il modello di progetto



Le classi ed i moduli client

- **Avatar**: rappresenta l'utente nell'ambiente virtuale, mantiene i dati relativi alla posizione attuale ed ai vicini;

Le classi ed i moduli client

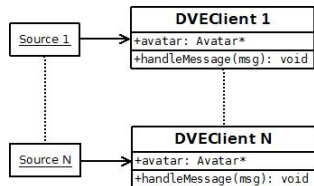
- **Avatar**: rappresenta l'utente nell'ambiente virtuale, mantiene i dati relativi alla posizione attuale ed ai vicini;
- **Source**: dalla libreria `queueinglib` simula le azioni dell'utente nell'utilizzo del client;

Le classi ed i moduli client

- **Avatar**: rappresenta l'utente nell'ambiente virtuale, mantiene i dati relativi alla posizione attuale ed ai vicini;
- **Source**: dalla libreria `queueinglib` simula le azioni dell'utente nell'utilizzo del client;
- **DVEClient**: è il client vero e proprio ed è definito dal modulo `DVEClient.ned` e dalla relativa classe C++ che ne definisce il comportamento.

Generazione di eventi

- Ogni client è rappresentato da due moduli **Source** e **DVEClient** connessi;
- Source genera dei messaggi ad intervalli regolari che vengono interpretati come azioni;
- la prima azione è il login, ed avviene ad un tempo casuale nei primi 10 secondi di simulazione.



omnetpp.ini

```
DVESystem.source[*].interArrivalTime = 2s
DVESystem.source[*].startTime = uniform(0s,10s)
```

Registrazione dati e statistiche

- **Risposta del sistema:** calcola il tempo simulato di risposta del sistema, ovvero ad ogni movimento del client misura il tempo impiegato per notificare tutti i client coinvolti.

Registrazione dati e statistiche

- **Risposta del sistema:** calcola il tempo simulato di risposta del sistema, ovvero ad ogni movimento del client misura il tempo impiegato per notificare tutti i client coinvolti.
- **Movimenti persi:** tiene conto dei movimenti persi di ogni client. Quando un nuovo job arriva al client per effettuare un movimento, se non è ancora arrivata la notifica (ACK) di quello precedente non è possibile effettuare il nuovo.

Registrazione dati e statistiche

- **Risposta del sistema:** calcola il tempo simulato di risposta del sistema, ovvero ad ogni movimento del client misura il tempo impiegato per notificare tutti i client coinvolti.
- **Movimenti persi:** tiene conto dei movimenti persi di ogni client. Quando un nuovo job arriva al client per effettuare un movimento, se non è ancora arrivata la notifica (ACK) di quello precedente non è possibile effettuare il nuovo.
- **Movimenti nulli:** conta i movimenti non effettuati “volutamente” dall'utente.

Registrazione dati e statistiche

- **Risposta del sistema:** calcola il tempo simulato di risposta del sistema, ovvero ad ogni movimento del client misura il tempo impiegato per notificare tutti i client coinvolti.
- **Movimenti persi:** tiene conto dei movimenti persi di ogni client. Quando un nuovo job arriva al client per effettuare un movimento, se non è ancora arrivata la notifica (ACK) di quello precedente non è possibile effettuare il nuovo.
- **Movimenti nulli:** conta i movimenti non effettuati “volutamente” dall'utente.
- **Movimenti:** conta i movimenti effettuati con successo.

Registrazione dati e statistiche

- **Risposta del sistema:** calcola il tempo simulato di risposta del sistema, ovvero ad ogni movimento del client misura il tempo impiegato per notificare tutti i client coinvolti.
- **Movimenti persi:** tiene conto dei movimenti persi di ogni client. Quando un nuovo job arriva al client per effettuare un movimento, se non è ancora arrivata la notifica (ACK) di quello precedente non è possibile effettuare il nuovo.
- **Movimenti nulli:** conta i movimenti non effettuati “volutamente” dall'utente.
- **Movimenti:** conta i movimenti effettuati con successo.
- **Presence Factor:** conta la “numerosità” della Aol, ovvero il numero di altri avatar nella medesima zona.

Registrazione dati e statistiche

Modulo ned

```
simple DVEClient
{
  parameters:
    @signal[sysResponse]( type="simtime_t" );
    @statistic[dveResponse](...);
    @signal[presenceFactor]( type="unsigned_int" );
    @statistic[clientPresenceFactor](...);
    ...
}
```

Classe DVEClient

```
// Statistics.
simtime_t timeRequest;
simsignal_t systemResponseSignal;
int ackReceived;
simsignal_t presenceFactorSignal;
unsigned int presenceFactor;
...
```

Registrazione dati e statistiche

Inizializzazione

```
...  
systemResponseSignal = registerSignal("sysResponse");  
presenceFactor = avatar->GetAoISize();  
WATCH(presenceFactor);  
presenceFactorSignal = registerSignal("presenceFactor");
```

Movimento

```
...  
avatar->move(x, y);  
send(move, "wanIO,o");  
timeRequest = simTime();
```

System Response

```
...  
simtime_t response = simTime() - timeRequest;  
emit(systemResponseSignal, response);  
// Movement complete: update presence factor.  
presenceFactor = avatar->GetAoISize();  
emit(presenceFactorSignal, presenceFactor);
```

Le classi ed i moduli server

- **VirtualAvatar**: rappresenta l'avatar all'interno dell'ambiente virtuale gestito dal Main Server;

Le classi ed i moduli server

- **VirtualAvatar**: rappresenta l'avatar all'interno dell'ambiente virtuale gestito dal Main Server;
- **VirtualEnvironment**: "simula" un'area in cui gli avatar possono muoversi liberamente e senza collisioni;

Le classi ed i moduli server

- **VirtualAvatar**: rappresenta l'avatar all'interno dell'ambiente virtuale gestito dal Main Server;
- **VirtualEnvironment**: "simula" un'area in cui gli avatar possono muoversi liberamente e senza collisioni;
- **MainServer**: è il server principale, composto dal modulo `MainServer.ned` e dalla relativa classe C++, si occupa di gestire l'ambiente virtuale e la partizione;

Le classi ed i moduli server

- **VirtualAvatar**: rappresenta l'avatar all'interno dell'ambiente virtuale gestito dal Main Server;
- **VirtualEnvironment**: "simula" un'area in cui gli avatar possono muoversi liberamente e senza collisioni;
- **MainServer**: è il server principale, composto dal modulo `MainServer.ned` e dalla relativa classe C++, si occupa di gestire l'ambiente virtuale e la partizione;
- **DVEServer**: sono i server di partizione, composti dal modulo `DVEServer.ned` e dalla relativa classe C++, si occupano di gestire le comunicazioni con i client.

Gestione del mondo

Quando un client notifica un movimento, il MainServer:

- Calcola la nuova Aoi del client e notifica i nuovi vicini;
- Aggiorna lo stato interno del mondo (VE).

```
int* newAoi = NULL;
unsigned int newAoiSize;
ve->GetAvatarAndSizeAt(x, y, &newAoi, newAoiSize);
UpdateAoiMsg* update = new UpdateAoiMsg();
update->setClientMoved(clientID);
update->setX(x);
update->setY(y);
update->setAoiArraySize(newAoiSize);
for (unsigned int index = 0; index < newAoiSize; index++)
{
    update->setAoi(index, newAoi[index]);
}
update->setIsNeighborNotification(false);
send(update, "IaiOut");
// Updates VA and VE.
VirtualAvatar* avatar = connectedAvatars_[clientID];
avatar->move(x, y);
```

Sistema di notifica

Quando un client notifica un movimento, il MainServer:

- Se non ci sono vicini coinvolti manda direttamente l'ack al client;
- Altrimenti crea un registro per raccogliere le notifiche dei vicini, l'ack sarà inoltrato in seguito.

```
int aksRequired = (msgAoiSize <= oldAoiSize) ? msgAoiSize + newAoiSize
                : newAoiSize + oldAoiSize;
if (aksRequired == 0)
{
    // No client to be notified, send the ack to client moved.
    ACKMsg* ack_msg = new ACKMsg();
    ack_msg->setMovedID(clientID);
    send(ack_msg, "lanOut");
}
else
{
    // Insert the new ack into the registry.
    Acknowledgment* ack = new Acknowledgment();
    ack->current = 0;
    ack->total = aksRequired;
    ack_registry_.insert(std::pair<int, Acknowledgment*>(clientID, ack));
}
```

Le classi ed i moduli della rete

- **WAN**: rappresenta la rete internet, è composta da un modulo `WAN.ned` e la relativa classe C++, si occupa di inoltrare e indirizzare i messaggi tra client e server;

Le classi ed i moduli della rete

- **WAN**: rappresenta la rete internet, è composta da un modulo `WAN.ned` e la relativa classe C++, si occupa di inoltrare e indirizzare i messaggi tra client e server;
- **LAN**: è un canale definito con una latenza molto bassa per simulare le connessioni della LAN, un anello unidirezionale.

Le classi ed i moduli della rete

- **WAN**: rappresenta la rete internet, è composta da un modulo `WAN.ned` e la relativa classe C++, si occupa di inoltrare e indirizzare i messaggi tra client e server;
- **LAN**: è un canale definito con una latenza molto bassa per simulare le connessioni della LAN, un anello unidirezionale.

Le classi ed i moduli della rete

- **WAN**: rappresenta la rete internet, è composta da un modulo `WAN.ned` e la relativa classe C++, si occupa di inoltrare e indirizzare i messaggi tra client e server;
- **LAN**: è un canale definito con una latenza molto bassa per simulare le connessioni della LAN, un anello unidirezionale.

DVESystem.ned

```
channel LAN extends ned.DelayChannel
{
    delay = 1ms;
}
```

Simulazione del delay

omnetpp.ini: definizione della media

```
DVESystem.wan.delayMean = 0.10 s
```

WAN.cc: inizializzazione della media

```
WAN::initialize()  
{  
    mean = par("delayMean");  
}
```

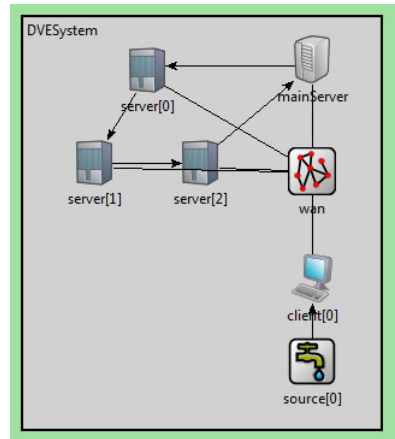
WAN.cc: tipico meccanismo di routing

```
const char* gateName = gate->getName();  
if (strcmp(gateName, "toClient$i") == 0)  
{  
    sendDelayed(l_msg, exponential(mean), "toMainServer$o");  
}  
else if (strcmp(gateName, "toMainServer$i") == 0)  
{  
    sendDelayed(l_msg, exponential(mean), "toClient$o", l_msg->getID());  
}
```


DVE System

Un esempio di DVE System

- 1 server principale;
- 3 server di partizione;
- n client ed i rispettivi moduli Source;
- una rete lan connessa ad ogni entità;
- i server connessi da un canale unidirezionale.



Simulazione

- La simulazione consiste in un'unica “run” del modello proposto;

Simulazione

- La simulazione consiste in un'unica “run” del modello proposto;
- 180 client che generano circa 80–90 response per un totale dell'ordine delle 15000–16000 variabili;

Simulazione

- La simulazione consiste in un'unica “run” del modello proposto;
- 180 client che generano circa 80–90 response per un totale dell'ordine delle 15000–16000 variabili;
- possibile utilizzare il metodo *Batch* per le stime statistiche, considerando ogni client un batch a sè stante.

Valutazioni

Valutare il DVE dal punto di vista della rete

Focus su tre parametri:

- `dveResponse`,
- `clientMovesLost`,
- `clientPresenceFactor`.

Valutazioni

Valutare il DVE dal punto di vista della rete

Focus su tre parametri:

- dveResponse,
- clientMovesLost,
- clientPresenceFactor.

Un primo risultato notevole

Il primo fatto che balza all'occhio analizzando i dati ottenuti è la totale **assenza di mosse perse** dai client.

Response Time

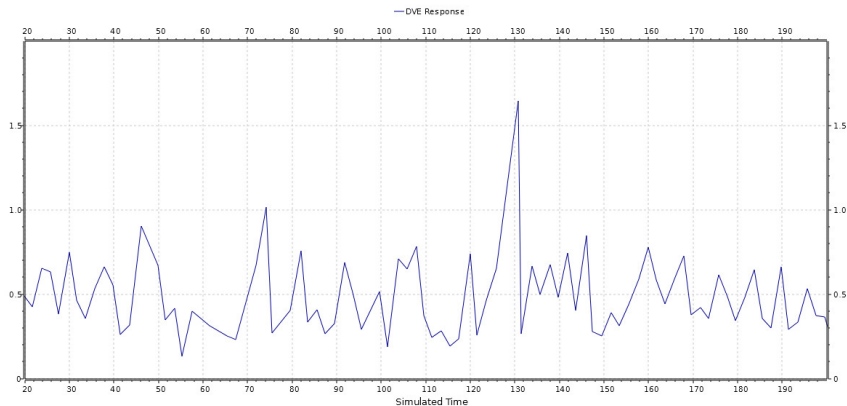


Figura : Il *Response Time* del client 175.

Analisi statistica

I dati vengono suddivisi in 180 batch, in ciascuno sono quindi presenti un numero n_i variabile di osservazioni ($i \in \{1, \dots, 180\}$). All'interno del batch i viene calcolata la media:

$$\bar{X}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} X_{ij}$$

Si calcola quindi la stima puntuale della media:

$$\hat{\mu} = \frac{1}{180} \sum_{i=1}^{180} \bar{X}_i$$

Ed infine la varianza campionaria:

$$S^2 = \frac{1}{179} \sum_{i=1}^{180} (\bar{X}_i - \hat{\mu})^2$$

Analisi statistica

Statistiche

Esportando i dati ottenuti dalla simulazione in formato CVS è stato possibile effettuare questi calcoli comodamente mediante uno script Python con i seguenti risultati:

- Stima puntuale della media $\hat{\mu} = 0.4993s$;
- Varianza campionaria $S^2 = 0.0008s$.

Analisi statistica

Statistiche

Esportando i dati ottenuti dalla simulazione in formato CVS è stato possibile effettuare questi calcoli comodamente mediante uno script Python con i seguenti risultati:

- Stima puntuale della media $\hat{\mu} = 0.4993s$;
- Varianza campionaria $S^2 = 0.0008s$.

Intervallo di confidenza

Con un livello di confidenza del 95% si ha il seguente intervallo di confidenza:

$$0.4928 \leq \hat{\mu} \leq 0.5058$$

Presence Factor

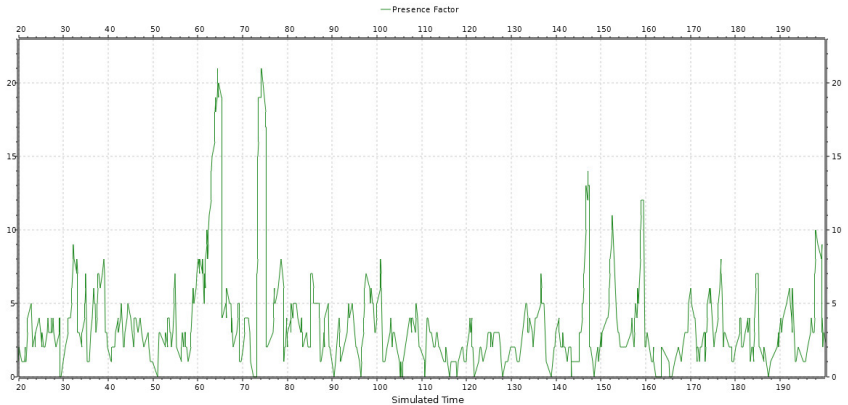


Figura : Il *Presence Factor* del client 175.

Presence Factor

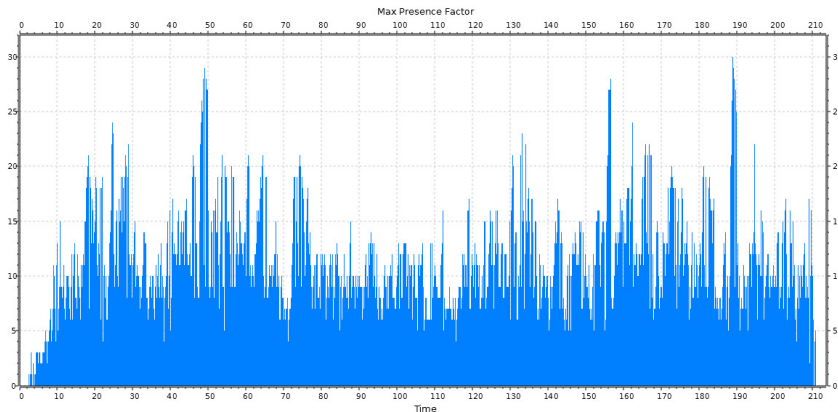


Figura : Massimo valore di *Presence Factor* dei client.

Relazione tra Response Time e Presence Factor

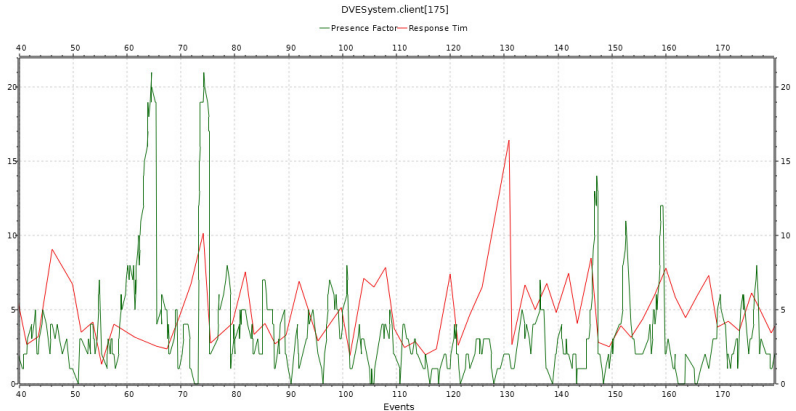


Figura : Mancanza di correlazione tra *Response Time* e *Presence Factor*.

Cosa accade aumentando gli utenti?

Statistiche

Con 3 server e 360 client si evidenziano i seguenti risultati:

- Stima puntuale della media $\hat{\mu} = 0.5405s$;
- Varianza campionaria $S^2 = 0.0024s$.

Cosa accade aumentando gli utenti?

Statistiche

Con 3 server e 360 client si evidenziano i seguenti risultati:

- Stima puntuale della media $\hat{\mu} = 0.5405s$;
- Varianza campionaria $S^2 = 0.0024s$.

Intervallo di confidenza

Con un livello di confidenza del 95% si ha il seguente intervallo di confidenza:

$$0.5354 \leq \hat{\mu} \leq 0.5456$$

Conclusioni e lavori futuri

- Studiati alcuni articoli sui DVE;

Conclusioni e lavori futuri

- Studiati alcuni articoli sui DVE;
- È stato creato un nuovo modello sulla base di quello proposto;

Conclusioni e lavori futuri

- Studiati alcuni articoli sui DVE;
- È stato creato un nuovo modello sulla base di quello proposto;
- Eseguite due simulazioni con 180 e 360 client coinvolti;

Conclusioni e lavori futuri

- Studiati alcuni articoli sui DVE;
- È stato creato un nuovo modello sulla base di quello proposto;
- Eseguite due simulazioni con 180 e 360 client coinvolti;
- L'analisi dei risultati ha evidenziato robustezza, non avendo ottenuto nessuna perdita di movimenti ed una buona efficienza;





Conclusioni e lavori futuri

- Studiati alcuni articoli sui DVE;
- È stato creato un nuovo modello sulla base di quello proposto;
- Eseguite due simulazioni con 180 e 360 client coinvolti;
- L'analisi dei risultati ha evidenziato robustezza, non avendo ottenuto nessuna perdita di movimenti ed una buona efficienza;
- In futuro si potrebbe espandere il modello;

Conclusioni e lavori futuri

- Studiati alcuni articoli sui DVE;
- È stato creato un nuovo modello sulla base di quello proposto;
- Eseguite due simulazioni con 180 e 360 client coinvolti;
- L'analisi dei risultati ha evidenziato robustezza, non avendo ottenuto nessuna perdita di movimenti ed una buona efficienza;
- In futuro si potrebbe espandere il modello;
- Analizzare più approfonditamente il sistema di partizionamento.

Riferimenti

-  P.Morillo, J.M.Orduna, M.Fernandez, and J.Duato. *Improving the Performance of Distributed Virtual Environment Systems*. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 16(7), 2005.
-  M. Zyda, *From Visual Simulation to Virtual Reality to Games*, Computer Society, September 2005;
-  S.A. van Houten, P.H.M. Jacobs, *An Architecture for Distributed Simulation Games*, Proceedings of the 2004 Winter Simulation Conference;
-  C. Ghosh, R.P. Wiegand, B. Goldiez, T.Dere, *An Architecture Supporting Large Scale MMOGs*, Proceedings of the 3rd International ICST Conference on Simulation Tools and Technique, 2010.

```

BATCH = 180

def main(args):
    x_bar = []

    # Compute mean
    for current in range(1, BATCH + 1):
        responseFile = 'responses-' + str(current) + '.csv'
        respReader = csv.reader(open('../data/' + responseFile), delimiter = ',')
        x_bar_i = 0
        n = 0
        for row in respReader:
            if row[0] != 'time':
                if float(row[0]) > 20.0 and float(row[0]) < 180.0:
                    x_bar_i += float(row[1])
                    n += 1
        x_bar.append(x_bar_i / n)
    mu = 0
    for x_i in x_bar:
        mu += x_i / BATCH
    print('Stima puntuale della media: ' + str(mu))

    sigma2 = 0
    for x_i in x_bar:
        sigma2 += (x_i - mu)**2
    sigma2 /= (BATCH - 1)
    print('Varianza campionaria: ' + str(sigma2))

    a = mu - 1.96 * (sqrt(sigma2) / sqrt(BATCH))
    b = mu + 1.96 * (sqrt(sigma2) / sqrt(BATCH))
    print('Intervallo di confidenza: (' + str(a) + ', ' + str(b) + ')')

```