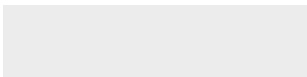


## MMOG Performance with the Versant Object Database

A Versant White Paper by Derek Laufenberg  
日本語訳 佐藤 剛宣(takenori@versant.com)

© *Versant Corporation*  
*255 Shoreline Drive, Suite 450*  
*Redwood City, CA 94065*  
*email: [info@versant.com](mailto:info@versant.com)*  
*Main: (650) 232 2400*  
*Fax: (650) 232 2401*  
*web: [www.versant.com](http://www.versant.com)*



## 概要

このテストは、詳細なゲームモデルとリアルタイム負荷計測によって、現実的な負荷をデータベースサーバーに発生させることができます。その時計測された性能データを見れば、テストに用いたサーバーの性能が分かると同時に、そのサーバーによって最大何人までプレイできるか予測できるように、「プレイヤー・分」という考えを導入しています。

一般的なPCハードウェア上で実施した250人プレイのテストでは、結果のタイミングデータを見ると実際の負荷の数百倍の性能が出ました。これはリアルタイムなレスポンスにまだまだ余裕があるということで、もっと多くのプレイヤーをサポートできるということを意味します。さらにそこから逆算して、7万人超のプレイヤーをサポートするのに必要なシステムも検討します。

**訳注：**このホワイトペーパーは、オンラインゲーム開発に詳しい方を対象にしています。そうでない場合は、オンラインゲームに必要なとされる条件を前提に読んでいただくと理解しやすくなります。その条件は、「刻々と変化する世界をリアルタイムに表現する」ということです。この場合、時間のかかるデータアクセスを待っていることができないため（ゲームが止まってしまう）、非同期やバッチの処理で帳尻を合わせる必要が出てきます。そうすると開発が一気に複雑になり、バグの温床となることが少なくありません。

## 説明

このテストで使用するゲームモデルは、キャラクターが豊富なWoWやEverQuestなどの人気オンラインゲームを参考にしています。そしてそのゲームモデルに対し、実際のゲームサーバーが行う様々なオペレーションを行います。テストクライアントはキャラクターの状態を変更させるようなアクションを行い、そこで変更または新たに作られたオブジェクトは直ちにデータベースサーバーに送られ、実行されます。このテストクライアントは、データベースがサポートできる最大プレイヤー数に達するまで、できるだけ速く実行されていきます。そしてこの時、各アクション毎にタイミングデータが集計されます。

「標準的なプレイヤー」を、ある一定時間内にある負荷をかけるものとして想定することは可能でしょう。そこでそのような標準化をするために、各ゲームアクティビティの「毎分あたりのプレイヤー」という指標を導入します。この「毎分あたりのプレイヤー」は、あるプレイヤーが1分間の間に行った全てのアクションの回数と定義します。各アクションの実行頻度は異なりますが、トータルで見ると、相当数のプレイヤーがかかるデータベースの負荷を代表するものになるはずです。また、1分間に何度も実行されるアクションもあれば、1時間に数回しか実行されないものもあるので、中には1以下の値もあります。

このモデルを使って、異なる負荷をかけることも可能です。必要に応じて負荷を変更し、データベースにどのような負荷がかかるかを確認できます。このペーパーの中では、ログインプロファイルと、アクティビティプロファイルを評価し

ています。アクティビティプロファイルは、次から次にアイテムの生成削除を行う行動的なプレイヤーを想定しています。

## ゲームモデル

下の図1は、今回C++で実装されたゲームモデルのUML図です。Playerとそれをサポートするクラスで、実際のゲームの実行をシミュレーションします。プレイヤーの所有するオブジェクトがアクションによって変更された場合、それはデータベースで反映されます。ほとんどのMMOGがそうであるように、プレイヤーのアイテム (Item) と職業 (Guild) が全てです。そのため、それらのコンセプトは詳細にモデル化されています。実際プレイヤーの負荷は、プレイヤーが所有するアイテムの数に大きく依存します。

プレイヤーがロードされる時には、そのプレイヤーが所有するアイテムも同時にロードされます。この時、全てのアイテムがテストクライアントにロードされると同時にサーバーでロックし、他のクライアントが変更できないようにします。こうしたロードとロックによって、MMOGで発生しやすいアイテムの重複バグを防ぐことができます。

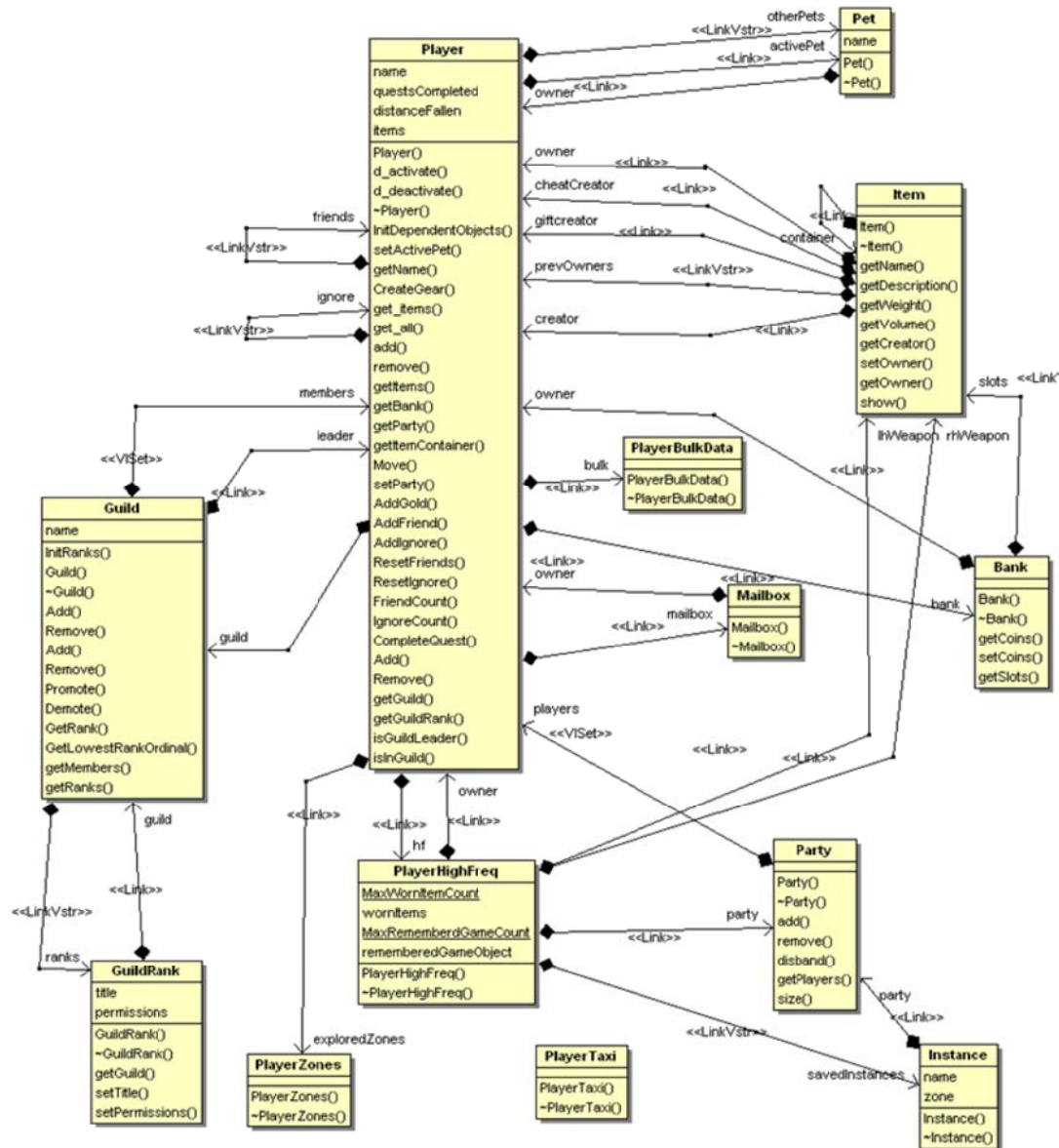


図1 ゲームモデル UML

このテストでは、各プレイヤーは100アイテムを所有した状態で生成されます。アクションによってアイテムは増加したり減少したりしますが、アイテムの生成が多くなるようになっていきますので、最終的には、各プレイヤーが200から300のアイテムを所有することになります。この時、各プレイヤーのサイズは約16KBになります。これは現在主流のMMOGのプレイヤーよりも相当詳細なもので、もっといろいろなやり取りが可能になります。

例えば今回のモデルでは、アイテムのやり取りの記録を残すようにしています。こうすることで、ソーシャルネットワーキングの機能を盛り込んだり、またはある種の詐欺行為を検出したりすることができるでしょう。

## テスト環境

今回のテストは、二台のマシンで実施しました。クライアントに WindowsXP ノート PC、サーバーには Linux デスクトップ PC を使用しました。また、2 台の通信には、専用の 1GB イーサネットを使用しました。各 PC の詳細なスペックは以下の通りです。

### クライアント

Intel T7200 Core2 Dual 2.0 GHz  
2 GB RAM  
Windows XP 32bit – SP3

### サーバー

Intel Q6600 Core2 Quad 2.4 GHz  
4GB RAM  
Linux 5 64bit

## テストデータベース生成

このテストデータベースには、アカウントが 5 万件生成されます。そして各アカウントは、5 プレイヤー、100 個のアイテムを所有します。また、このデータベースには五千の職業 (Guild) が設定され、ランダムにプレイヤーが割り当てられます。その結果、今回のテストでは、データベースサイズは二つのファイルボリュームにまたがって、8GB になりました。ちなみに、このデータベースは、SATA 7200 回転の一般的なディスク上で生成しました。

### Database Population

50000 instances of 'Account'  
250000 instances of 'Player'  
250000 instances of 'PlayerTaxi'  
250000 instances of 'PlayerZones'  
250000 instances of 'PlayerBulkData'  
250000 instances of 'PlayerHighFreq'  
250000 instances of 'Mailbox'  
250000 instances of 'Bank'  
250000 instances of 'Pet'  
250000 instances of 'VISet<Quest>'  
25049127 instances of 'Item'  
5000 instances of 'Guild'  
30000 instances of 'GuildRank'  
5000 instances of 'Quest'  
10004 instances of 'ItemDescriptor'

## ログインテストケース

ログインとキャラクターの取得プロセスは、別のサーバー上で、時には別のデータベース上で行われることがよくあります。この場合でも、Versant では異なるデータベース上のオブジェクトを意識することなく参照することが可能です。このテストでは、ログインプロセスに絞って (クエリ、キャラクターの取得、アカウント情報の更新)、この状況を再現しています。

ログインが最初に実行するクエリは、生成された五万のアカウントに該当するユーザー名をランダムに決定して実行します。そうして取得されたユーザーのア

カウントオブジェクトは、クライアントにロードされて更新された後、サーバーへ送られます。この時、そのアカウントのプレイヤーが持つキャラクターの論理オブジェクト ID も一緒にサーバーに送られ、そのキャラクターがロードされるようにします。これはかなり一般的なプロセスだと言えます。このテストではプレイヤーは実際にはロードされないため、ログイン処理に必要なシステムリソースを評価しました。

このテストの中では、一連のテストクライアントが次々にログインタスクを生成し、それが実行されるに従ってサーバー上に負荷がかかります。各テストクライアントはログインを一定時間内に千回実行し、ログインの合計数と合計所要時間を記録します。この時の CPU の利用率は Windows 上ではタスクマネージャー、Linux 上では `iostat` を用いて計測しました。

### ログインテスト結果

ログインを実行するテストクライアントの数は CPU の利用率が 80% になるまで増加されました。今回のテストでは、この時 8 クライアントが実行されました。

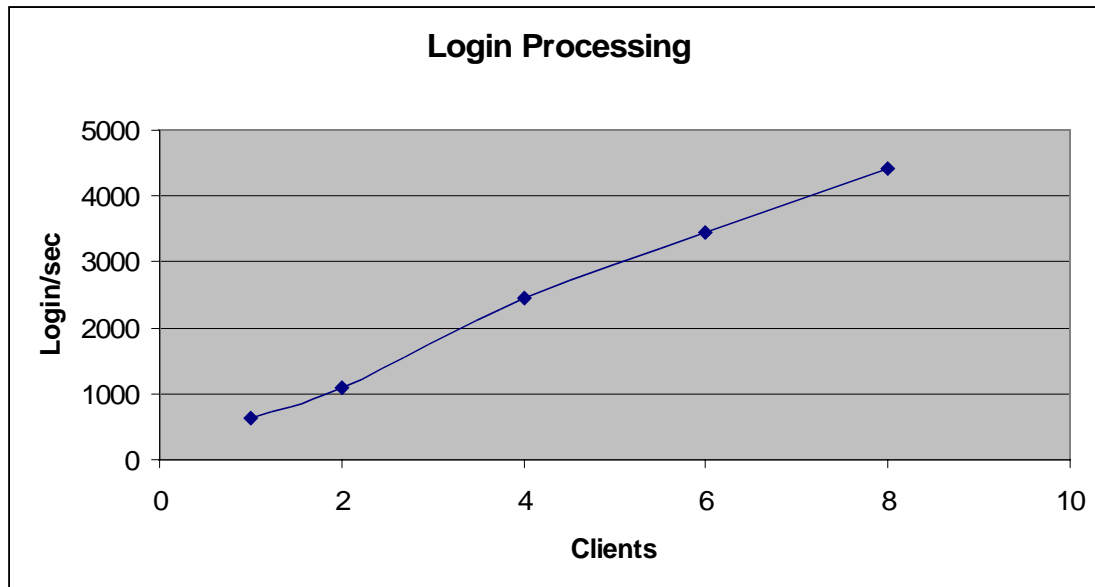


図2 ログイン/秒

このログイン処理には、アカウント情報の更新が含まれます。つまり新規生成はありませんが、クエリと読み込み、そして更新処理がデータベースに対して実行されます。しかしこの更新データはごく僅かですので、CPU の性能に大きく依存していることが分かります。

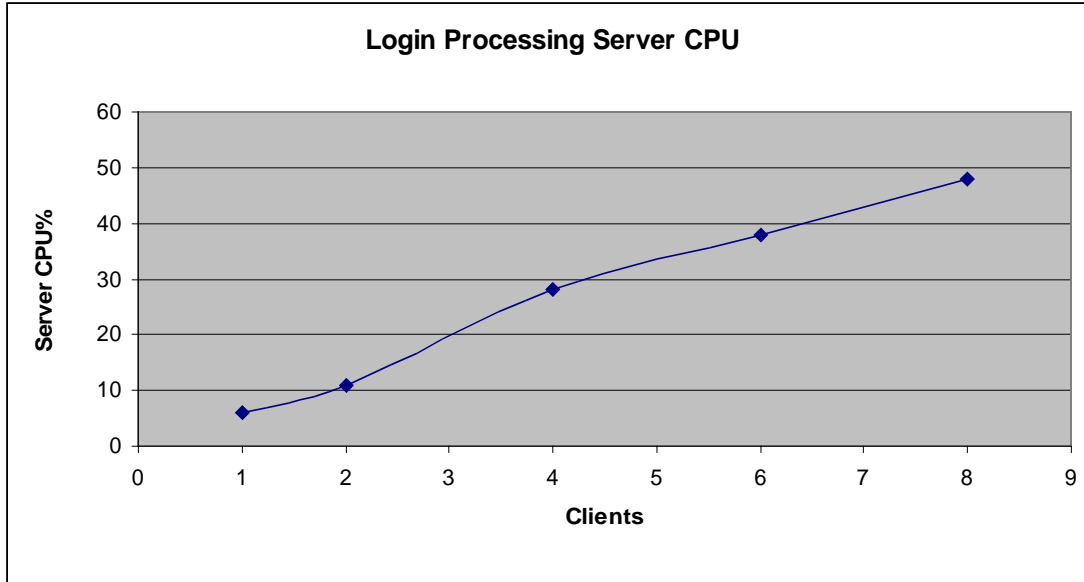


図 3 Database Server CPU Utilization

今回のテストで使用したごく一般的なデスクトップマシン上でも、四千ログイン/秒が達成されました。その時のCPUの利用率を見てみると、線形に増加して最大約50%の利用率でした。四千ログイン/秒というのは、どのゲームで考えても十分な性能だと言えます。そしてこれが、ごく一般的なデスクトップマシン上のVersantサーバーで実現できています。

### ゲームオペレーションテスト

このテストは、一連のアクションを実行することで、ゲームサーバーから送られてくる実際のプレイヤーによる負荷をシミュレートします。このテストにプレイヤーの上限を与えることで、各アクションの実行頻度やそれを実行するプレイヤーが決定されます。そしてそれらのアクションがテストの間に何度も実行されます。テストは、1分間相当のテストの繰り返しですが、実際の実行時間が1分よりも短い場合、結果的に実際よりも短い時間でテストが実行されることとなります。

アクションの実行回数は、単純に1分間に何回そのアクションを実行するかの予測に基づいています。実際のシステムでは、プレイヤーの状態とデータベースの同期を1分毎に行うようにすることが多いので、これは現実的な想定だといえます。

テスト実行中には、アクションが次々に実行され、システムの性能が測定されていきます。下のテーブルは、250プレイヤーを設定した時のアクションとその回数です。この設定は、データベースのストレスを大きくするために、相当アクティブなプレイヤーに設定しています。

Action	Counts/ Period	Description
Login	1	Random Account queried, player selected
LoadPlayer	1	Player graph and items loaded
Money Update	500	Player object updated: money, exp., position
GuildRoster Read	2	Guild, ranks and members loaded
Join Guild	1	Player leaves one guild, joins another
Create Item	250	New Item object created and put in inventory
Write List	1	Changed objects pushed to DB server
Delete Item	200	Player item is deleted
Update Item	500	Item is changed and updated
Item Stack Size	500	Item's count is changed (similar to Update)
Complete Quest	250	New quest object created for player
Friend	25	Player's Friend list is changed
Ignore	25	Player's Ignore list is changed
Eject Player	1	Player leaves simulation
Eject List	1	Ejected objects sent to DB, then released from memory
Checkpoint	1	Any other new/dirty objects pushed to DB

実際のゲームの想定に応じて、これらの設定は柔軟に変更することができます。これは250プレイヤー設定、つまり1つのゾーンで250プレイヤーがこれらのアクションを1分間に実行する設定です。

プレイヤーはランダムに抽出されますので、ベンチマーク用にディスクアクセスが効率化されることもありません。

この設定では、アイテムの生成、削除、更新をかなり頻繁に行います。これだけ見ると、250プレイヤーの設定を大幅に超えているといえるかもしれません。ちなみにこの設定だと、1分間相当内に、アイテムの変更が1000回、生成が250回、削除が200回行われます。

テーブルに挙げたアクションのうち、いくつかは純粋にデータベース用のものです。具体的に言うと、WriteList、EjectList、Checkpointです。これらのアクションは変更をサーバーへ送るためのもので、結果的にネットワークとサーバーでのファイルIOを引き起こすので、ほとんどの所要時間はこれらに費やされます。Checkpointは一定間隔で全てのゲームオブジェクトとデータベースを同期させるためのもので、更新又は新規生成されたオブジェクトでまだデータベースに書き込まれていないものがデータベースへ送られます。

EjectListはクライアント内のメモリ管理に必要なもので、シミュレーションを終了したプレイヤーのメモリーを開放します。一方、LoginとLoadPlayerは、シミュレーションに新しいプレイヤーを追加するためのものです。

## ゲームテストの結果

今回のテストでは、最大で1700分相当時間のテストを実行しました。そこで得られたデータが以下の図4になります。250プレイヤーで29時間相当のテストを、わずか5分で実行したことになります。

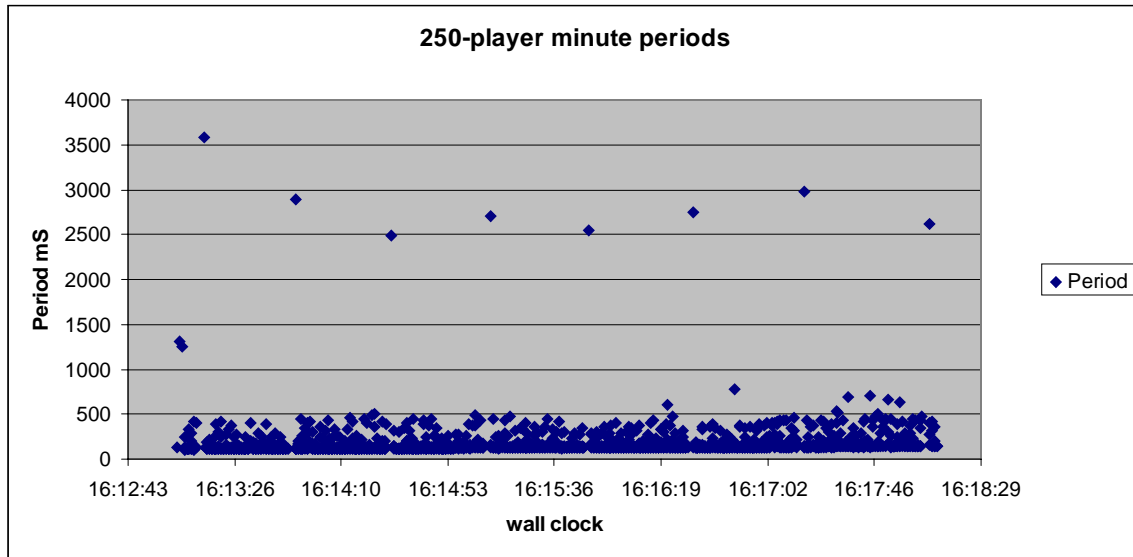


図 4 Timing Samples for a 250 Player-Minute Load

これを見ると、1分間相当のテストの平均所要時間は176ms、言い換えると、1分間に340分相当のテストを実行したことになります。つまり想定される実際の負荷を340倍速く処理していることになります。

サーバーの統計を見てみると、iostatは概ね30から60%の利用率を示しており、3-50msの待ち時間になっていました。一方CPUは40から50%の利用率で推移していました。

クライアントで利用したネットワークの帯域を見てみると、約2.5%となりました。ちなみにクライアントとサーバー間には、専用の1Gbイーサネット接続を使用しました。

図4を見ると、定期的に平均処理時間を大幅に上回っているのが分かります。この時データベースでは、トランザクションとアンドゥログの処理が行われていて、どちらかのファイルサイズが最大になり、新しいファイルを生成する時に発生します。バックグラウンドスレッドで処理される場合もありますが、今回のテストのように大きなサイズのログ\*を使用した場合、数秒かかる場合もあります。  
\* Versantのデータベースフォルダ内のprofile.beにあるphysical.logとlogical.logを参照

このチェックポイント処理には数秒かかるかもしれませんが、これらの処理を含んだ結果だということを忘れないでください。ちなみにゲームサーバーでは通常こうした処理はバックグラウンドで処理されますので、時々発生するこうした遅延が吸収されるようになっています。

このテストでは、64MBのphysical.log（アンドゥ）とlogical.log（トランザクション）を使用しました。これらの処理が重い場合、ログファイルを別のディスク

クに移すと性能が向上します。また、サイズを大きくすることで、この処理の回数を減らし、トータルの性能を向上させることができます。また、ログファイルをディスクに同期させるタイミングをチューニングすることで、より細かい性能改善も得られるでしょう。

## Two Client Operation Test

ゲームオペレーションテストではシステムリソースが有り余っていたので、さらにもう1つのクライアントを実行してみました。同じように250プレイヤーですが、結果はほとんど同じでした。図の5と6がその結果です。

1分間相当のアクションの平均実行時間は、両方とも420msでした。正確に言うと、クライアント1が417msで、クライアント2が424msでした。これは毎分280分相当の負荷をこなせることになりますが、先ほどのテスト結果の340よりかなり少なくなりました。この主な原因は、サーバー上のディスクの大域幅不足によるものです。

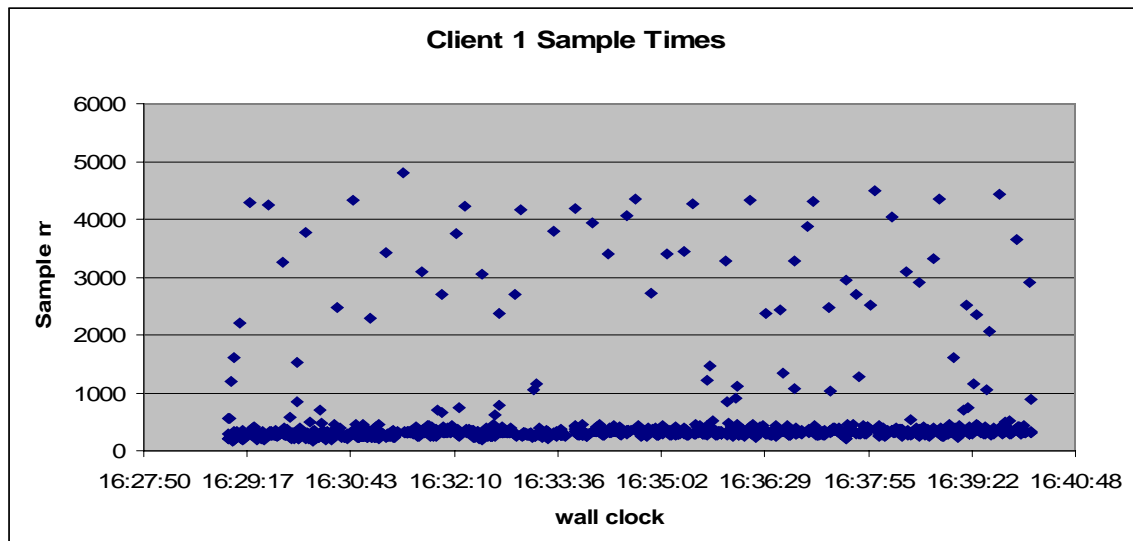


図 5 Timing Samples for a 500 Player-Minute Load – Client #1

サーバー上で `iostat` を見ると、ディスクアクセスがチェックポイントでフル稼働になっているのが分かりました。これは結果的にディスクアクセスの待ち時間の増加につながります。この2クライアントのテストでは、通常の待ち時間は10から200msでしたが、チェックポイントでは700msにも達していました。こうした待ち時間の増加と、2クライアントによるチェックポイント要求が2倍になったことで、結果的に大きな性能劣化になりました。このことから、ディスクが性能の上限に大きく影響することがわかります。ちなみにこの時のCPUの負荷は65から75パーセントで、まだ余裕がありました。

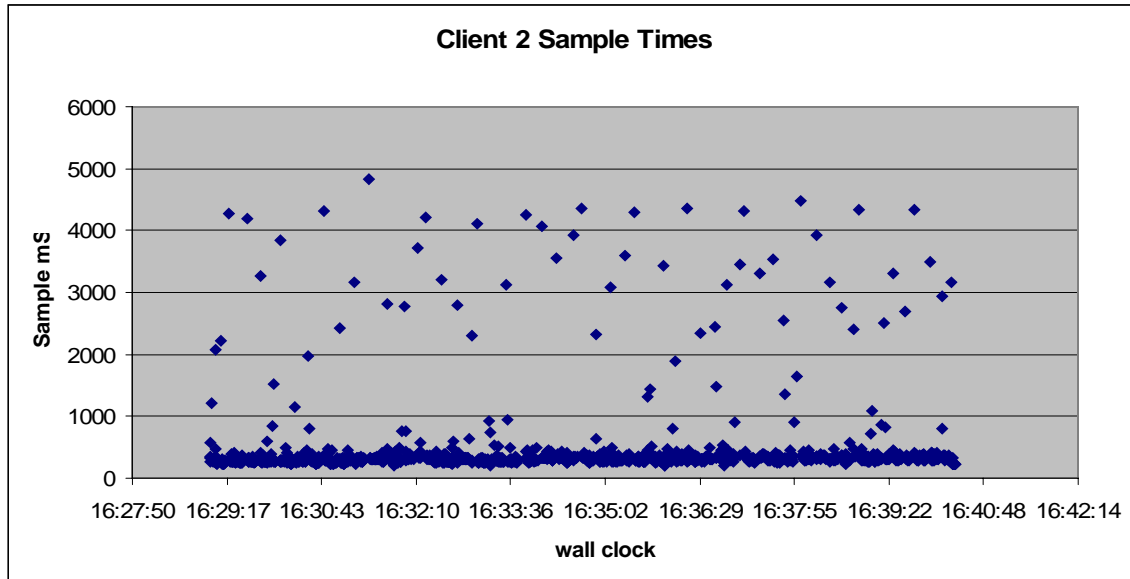


図 6 Timing Samples for a 500 Player-Minute Load – Client #2

2クライアントとも、ほとんど同じような結果でした。また、チェックポイントにより時間がかかりました。しかし、500プレイヤーの1分あたりの負荷を、わずか425msで処理していることを忘れないでください。

## 結果

これらのテストから、250プレイヤーをサポートする負荷を、280から340倍速く処理できることが分かりました。実際のゲームサーバーでは、もっと少ない負荷になるでしょう。つまり250プレイヤーよりもっと多くのプレイヤーを、このテストで見たよりも少ないシステムリソース消費で実行できることとなります。標準化されたデータと毎分あたりのプレイヤーという考えで見ると、1台のVersantデータベースサーバー上で7万から8万5千プレイヤーをサポートすることになります。もちろんその時はもっと多くのメモリが必要になりますが、そのコストは大きなものではありません。

このような人口的に負荷をかけるのはまだまだですが、システムにゲームシナリオがどう負荷を与えるかを理解する手助けにはなります。今回用いたプレイヤーのアクションをプロトタイプする方法は、ゲームシステムによくある想定外のトラブルなどの解決に効果的でしょう。つまり開発の早い時点から、どのようなハードウェアが必要か、フィードバックを得ることができます。今回使用したゲームモデルは、Versantが改善していきますが、今回のようなテストによって、検証していくことができます。

Versantは、こうしたサーバーへのオブジェクトの効率的な転送やサーバーでのオブジェクトキャッシングにおいてユニークな特徴を持っているため、MMOGのような高性能が要求されるシステムに最適です。

## Appendix A – サンプルデータ

以下のサンプルは、250 プレイヤーのゲームテストの出力例です。各行は1分あたりの250 プレイヤーのアクションを実行するのに要した時間を表しています。また100分相当毎に、それぞれのアクションの詳細な性能データも出力されます。

```

16:13:19 T_last = 109ms
16:13:19 T_last = 113ms
16:13:19 T_last = 126ms
16:13:20 T_last = 145ms
16:13:20 T_last = 301ms
16:13:20 T_last = 118ms
16:13:20 T_last = 120ms
16:13:20 T_last = 113ms
16:13:20 T_last = 120ms
16:13:20 T_last = 121ms
16:13:21 T_last = 416ms
16:13:21 T_last = 115ms
16:13:21 T_last = 248ms
16:13:21 T_last = 115ms
16:13:21 T_last = 112ms
16:13:22 T_last = 111ms
16:13:22 T_last = 114ms
16:13:22 T_last = 116ms
16:13:22 T_last = 118ms
16:13:22 T_last = 113ms
16:13:22 T_last = 116ms
16:13:22 T_last = 120ms
16:13:22 T_last = 113ms
16:13:22 T_last = 111ms

```

NAME	C/per	CALLS	T.Time	TT/calls	mean	Tn ms	Tn-1 ms
Login	1	100	2mS	0.02	0.03	0.01	0.01
LoadPlayer	1	100	255mS	2.55	2.56	2.33	2.42
MoneyUpdate	500	50000	202mS	0.00	0.01	0.00	0.00
GuildRoster	2	200	5012mS	25.06	25.08	17.97	18.72
Join Guild	1	100	738mS	7.38	7.39	1.24	1.15
CreateItem	250	25000	263mS	0.01	0.02	0.01	0.01
WriteList	1	100	2433mS	24.33	24.34	9.81	9.79
DeleteItem	200	20000	181mS	0.01	0.01	0.01	0.01
UpdateItem	500	50000	182mS	0.00	0.01	0.00	0.00
ItemStkSize	500	50000	180mS	0.00	0.00	0.00	0.00
EjectList	1	100	4394mS	43.94	43.95	194.35	28.31
CompleQuest	250	25000	1424mS	0.06	0.68	0.00	0.01
Friend	25	2500	7mS	0.00	0.00	0.00	0.00
Ignore	25	2500	9mS	0.00	0.00	0.00	0.00
EjectPlayer	1	3	0mS	0.03	0.03	0.03	0.03
EjectList	1	100	38mS	0.39	0.39	0.00	0.00
ChkPt	1	100	6344mS	63.44	63.45	16.47	15.58
Tmean	tn	tn-1					
379.17	270.32	105.60					

Periods= 100 Players= 100 Items =10970 GuildLoaded= 200  
TotalTime=21.66S avg Tau= 216 mS Logins/sec=4.62 Player/sec=4.62

```

16:13:23 T_last = 279ms
16:13:23 T_last = 111ms
16:13:23 T_last = 112ms
16:13:23 T_last = 153ms
16:13:23 T_last = 116ms
16:13:23 T_last = 131ms

```