

ベンチマークレポート

ーデータグリッド XWS 編ー

平成 22 年 9 月

グリッド協議会 先端金融テクノロジー研究会 ベンチマーク WG
日本アイ・ビー・エム 山本 学

《 目 次 》

1.	WXS: WEBSHERE EXTREME SCALE (IBM)	1
1.1	WXS: WebSphere eXtreme Scale の機能概要.....	1
1.1.1	概要.....	1
1.1.2	適用事例.....	1
1.1.3	機能説明.....	2
1.2	WXS: WebSphere eXtreme Scale の評価結果.....	5
1.2.1	ベンチマーク実行環境.....	5
1.2.2	評価シナリオ1:レスポンスタイムからみたスケーラビリティ.....	6
1.2.3	評価シナリオ2:スループットからみたスケーラビリティ.....	8
1.2.4	評価結果.....	9
1.2.5	考察.....	9

1. WXS: WebSphere eXtreme Scale (IBM)

1.1 WXS: WebSphere eXtreme Scale の機能概要

1.1.1 概要

WebSphere eXtreme Scale(以下、WXS)は、IBM 社が提供するデータグリッド製品である。IBM は、キーバリューストア型のデータグリッドと、インメモリ RDB をクラスタ化させたインメモリ RDB 型のデータグリッドの両方の技術を持つが、WXS は、キーバリューストア型のデータグリッド製品である。WXS は、多数のサーバによるスケールアウト構成を基本アーキテクチャとしており、主に証券やクレジットカード、オークションなどの高速処理と高スケーラビリティが必要とされる場所で使用されている。

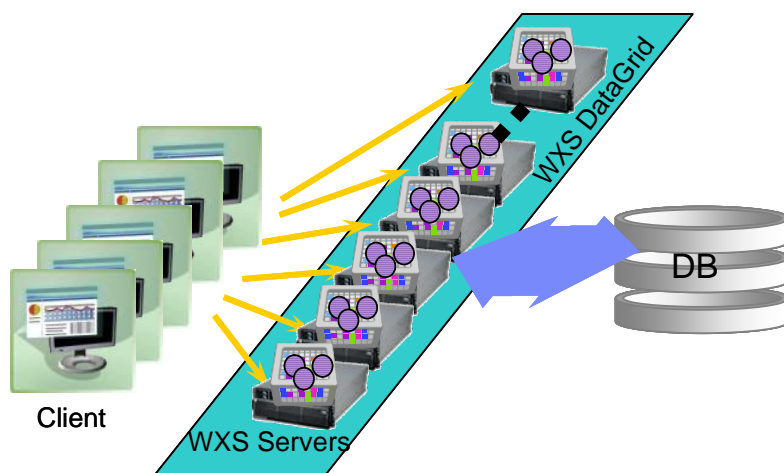


図 1-1 IBM WebSphere eXtreme Scale の概要

図 1-1 では、WXS を独立したデータグリッドとして構築した場合の概要を示している。WXS は、クライアントの Java Virtual Machine (以下、JVM) と同一プロセスおよび別プロセスの双方に組み込むことが可能である。例えば、WXS のクライアントとサーバを IBM WebSphere Application Server の同一 JVM 内で稼働させることができる。この場合、通信を介さずにデータ取得が可能となり、速いレスポンスが重要なアプリケーションで効果を発揮できる。

1.1.2 適用事例

(1) 米国クレジットカード会社のコールセンターにおける顧客情報取得の高速化の事例

この会社では、コールセンター内にてコールテイカーが顧客情報を取得する際に、顧客データベースに負荷が集中し、速やかに顧客情報を取得できないという課題があった。コールセンターでは、コールテイカーの待ち時間は無駄な時間であり、そのような時間をなくすことはコスト削減に直結することである。この課題を解決するために、この会社では、WXS をアプリケーションとデータベースの間に配置し、顧客データをキャッシュすることで、応答時間の短縮を実現させた。この会社では、エンタープライズサービスバス(以下、ESB)がアプリケーションとデータベースの間に配置されていたため、顧客データ取得においては ESB のメディエーションから WXS に Cache されている顧客データにアクセスさせる方法を用いた。これにより、アプリケーションの改変なしに WXS による高速化が可能となった。

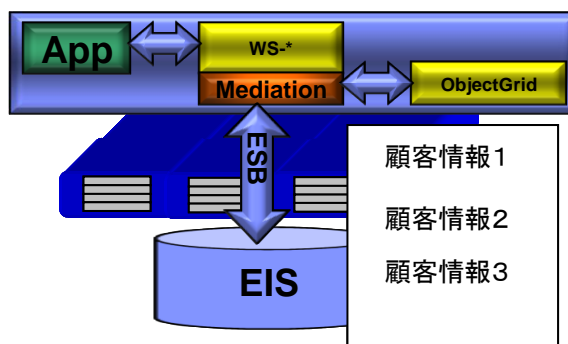


図 1-2 米国クレジットカード会社の事例

(2) 国内におけるスケーラブルなオークションシステムの事例

このケースは特定の企業が利用するオークションシステムにWXSを適用した例である。このシステムでは、スケーラビリティが重要な課題であった。このオークションシステムはその企業にとって新規ビジネスと位置づけられており、サービスイン直後のアクセス数はそれほど多くないものの、数年以内にアクセス数が劇的に増加することが見込まれていた。したがって、初期段階では少ないサーバで開始し、アクセス数増加に伴いサーバ追加で性能向上が可能となるシステムが重要な要件であった。この要件の実現にあたり、スケーラビリティを特徴とするWXSをWebSphere Application Serverに組み込み、オークションデータベース内のデータをキャッシュさせ、顧客要件を実現した。

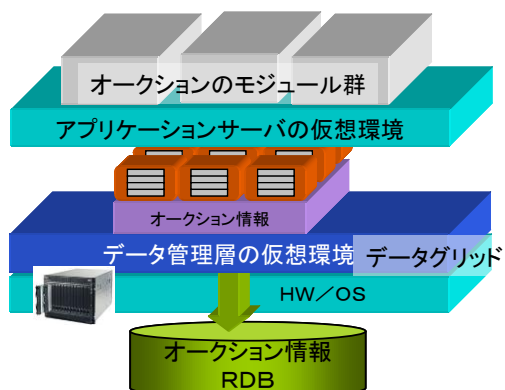


図 1-3 国内オークションシステムでの事例

1.1.3 機能説明

(1) 分散キャッシュ機能

WXS は、キーバリューストア型データグリッドである。キーバリューストアの基本機能は、キーを指定して、そのキーに対応するJavaオブジェクトをひとつ取得もしくは格納するというものである。アプリケーションから見たときにデータを格納する領域をオブジェクトマップと呼ぶ。オブジェクトマップは、データの種類ごとに定義され、WXSサーバがオブジェクトマップのサービスを提供する。キャッシュされるオブジェクトは、複数のWXSサーバ上で分散配置される。即ちオブジェクトマップが複数のサーバをまたがって存在することができるということである。

WXS のデータ分散は、完全パーティショニング手法を採用している。これは次のようなものである。WXS では個々のオブジェクトマップを管理するのではなく、アプリケーションからみて一つにみえるオブジェクトマップを

「パーティション」という区画に分けて管理する。パーティションは、JVM の単位ではなく、JVM 数とは独立して定めることができる。通常の利用形態では、ひとつの JVM 上で稼動する WXS サーバが複数のパーティションを提供するという構成である。

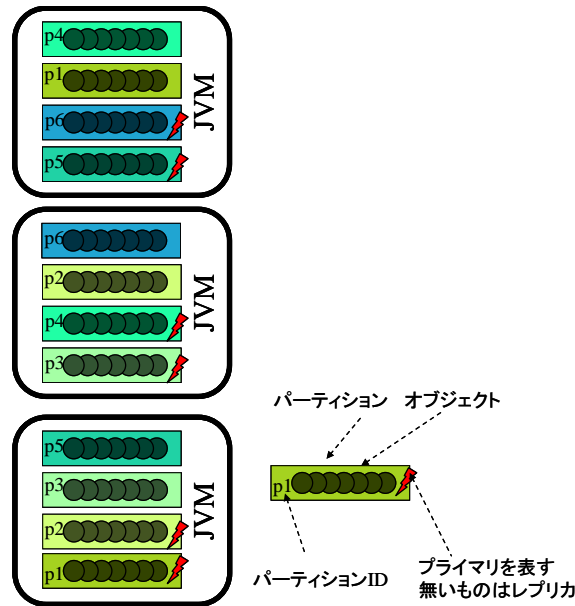


図 1-4 WXS のデータパーティショニングモデル

オブジェクトのキーからパーティションを特定する方法は、キーからハッシュ関数を用いて、パーティション番号を計算し、そのパーティション番号から WXS のサーバ番号を決定するという方法である。パーティション番号とサーバ番号のマッピングテーブルは、WXS の実行環境が管理しており、各サーバでこのテーブルのキャッシュを保持する。データグリッド内の WXS サーバ構成は、サーバ追加、サーバ保守のための計画停止、障害発生などに応じて動的に変化するため、WXS は、このマッピングテーブルを一貫して管理する機構を持っており、全サーバで同じテーブルが参照される仕組みを保持している。また、WXS にアクセスするクライアントランタイムも、このテーブルをキャッシュする。これにより、クライアントはキーからハッシュ関数とキャッシュされているマッピングテーブルを用いて、該当するオブジェクトが存在するサーバを即座に特定し、そのサーバに要求を発行する。即ち WXS はあるオブジェクトを取得するときに、WXS サーバ間でオブジェクトを探索したり、要求を転送するような通信は一切発生せず、サーバ数に応じた高いスケーラビリティを提供できる。

クライアントが保持するマッピングテーブルのキャッシュは、バージョン番号で管理されており、オブジェクトへのアクセス要求には、必ずそのバージョン番号が内包される。これにより、クライアントが古いマッピングテーブルでサーバにアクセスするケースにおいて、サーバ側でバージョン番号の不一致を検知することができる。この場合サーバはクライアントに新しいマッピングテーブルを送信し、クライアントは更新された最新のマッピングテーブルを基に再度アクセスを行う。

サーバ間でのマッピングテーブルもバージョンで管理されている。WXS は、その下位機構として、「HAManager」と呼ぶ多数のサーバを管理できる分散合意プロトコルに基づく障害検知機構を持つ。これは、IBM WebSphere Application Server Network Deployment が持つ障害検知機構と同じコンポーネントである。HAManager は、クラスタ内の構成が変化した場合、分散合意プロトコルを使って、クラスタ内の新しい構成の合

意を形成する。この合意形成に基づいて、マッピングテーブルが更新・伝播される。これにより全サーバで同一のマッピングテーブルを共有することができる。

WXS はトランザクション機能を提供しており、ひとつのトランザクションで複数のマップのエントリへの一括処理が可能である。ただし、ひとつのトランザクションで複数のエントリを更新する場合、それらが同じパーティションに属している必要がある。WXS は、各オブジェクトに対するロック機能も持っており、データ一貫性を保証することができる。

(2) レプリケーション機能

WXS は、同期レプリケーションと非同期レプリケーション機能を提供している。レプリカは、JVM 単位ではなく、パーティション単位である。これにより、ひとつの JVM には、複数のプライマリのパーティションと複数のレプリカのパーティションが存在することが可能であり、アプリケーション要件に応じた最適なレプリケーションを行うことが可能となる。

同期レプリケーション機能は、トランザクションのコミット時に、そのトランザクションが更新したオブジェクト群をレプリカ用のサーバにコピーするというものである。レプリカは、JVM 単位ではなく、パーティション単位である。コミットは、このコピーが完了するまでブロックされ、コピーが成功してクライアントに制御が戻る。

非同期レプリケーションは、トランザクションのコミット処理完了後の適当な時期に、非同期レプリカサーバに更新データを転送する。非同期レプリケーションのメリットは、コミット処理の時間を短くできる点にある。一方、更新データは、遅延してコピーされるため、あるデータのコミット完了後でコピーされるまでの間に、そのサーバが障害で停止すると、更新データが失われる可能性もある。

(3) グリッドエージェント機能

WXS は、データグリッド上のサーバで並列処理を行うためのグリッドエージェントフレームワークを提供している。クライアントで生成されたグリッドエージェントは複数のパーティションに転送され、各パーティションで並行処理された後にそれぞれのパーティションからクライアントに結果が転送される。クライアントは全ての結果を集約することができ、これはいわゆる Map-Reduce 的な処理モデルを実現する機能である。

(4) 検索機能

WXS の大きな特徴に、強力な検索機能がある。WXS は、オブジェクトに対する検索言語(RDB における SQL に類似)を提供しており、アプリケーションは、その検索言語で条件を記述して検索できる。またデータベースと同様にパフォーマンス向上を目的としたインデックスを生成することも可能である。

(5) リレーション定義

WXS には、エンティティマネージャと呼ぶ機能がある。この機能を使うと、キャッシュされるオブジェクトのスキーマを定義し、その定義において、他のオブジェクトへのリレーションを定義することができる。リレーションには、1対1、1対多、多対1、多対多の定義が可能である。このようなリレーションを定義されたデータ集合に対して、あるオブジェクトからリレーション先のオブジェクトの取得が可能となる。さらに、JOIN と同様の検索を行うことができる。

1.2 WXS: WebSphere eXtreme Scale の評価結果

1.2.1 ベンチマーク実行環境

ここでは、WXS を使用した各シナリオの性能計測結果をまとめる。使用した WXS ならびに JDK を下記に示す。

- WebSphere eXtreme Scale v6.1
- IBM JDK 1.6 Linux x64

本評価は、多数のクライアントからの同時アクセスを想定した場合の処理時間・スループットの評価である。例えば、1000 クライアントの場合の処理時間である。しかしながら、計測環境では、1000 台の物理ノードを準備できず、最大で 10 台である。そこで、図 1-5 に示す計測環境を構築して評価した。

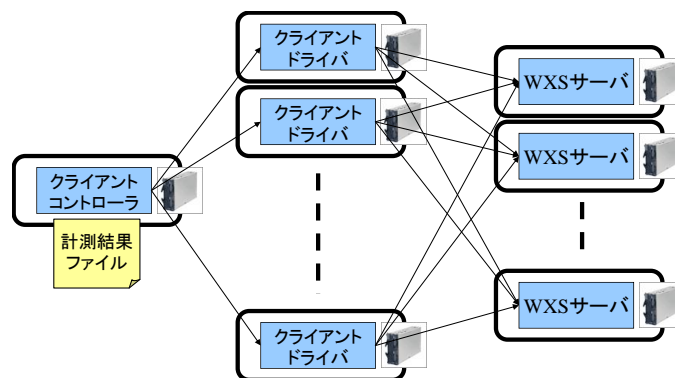


図 1-5 計測環境

WXS サーバの構成は、各パーティションにおいて同期レプリカ数を 1 つ持つという構成で、パーティション数は 16 とする。つまり、WXS サーバ数が 8 サーバの場合は、各 WXS サーバが持つパーティション数は、プライマリパーティションが 2、レプリカパーティションが 2 となる。

クライアントドライバがクライアントを表すプログラムであり、JVM である。ひとつのクライアントは、クライアントドライバのひとつのスレッドとして表される。クライアントコントローラは、全クライアントドライバに一斉に計測開始を指示する。各クライアントドライバは、この指示を受けて、データグリッドにアクセスし、一回 (1000 回の put/get) の計測完了ごとに計測結果をクライアントコントローラに送信し、次の計測を開始する。この通信は 1000 回の put/get に比べ非常に短いので、計測のオーバーヘッドにはならない。クライアントコントローラは、各クライアントドライバから受信した計測結果をファイルに書き出す。

各クライアントドライバでの計測は、2 つのシナリオである「スケーラビリティ評価」ならびに「スループット評価」共に、1000 個のオブジェクトの get/put に要した時間を計測するものである。一回の処理とは、1000 個のオブジェクトへ順次アクセスするものであり、その開始時刻と終了時刻を System.currentTimeMillis() で取得した値を計測結果とする。単位は、ミリ秒である。

最終的な計測結果は、各クライアントドライバにおいて、数十回の処理を繰り返し、すべてのクライアントドライバの全ての計測結果の平均値を計測結果とする。ただし、最初の数回は、ウォームアップ期間とし、計測値に含めない。

1.2.2 評価シナリオ 1 : レスポンスタイムからみたスケーラビリティ

(1) 計測方式

本シナリオでは、クライアント数が増加したときのレスポンスタイムを評価する。クライアント数は、1、100、1000 で変化させる。100 の場合のクライアント用のノード数は 10、1000 の場合は 20 であり、JVM 数も各ノードで 1 つずつとする。

(2) 測定結果

クライアント数を 100、1000 にした時のサーバ数の増加に対する処理時間の短縮度合いを図 1-6 から図 1-9 に示す。

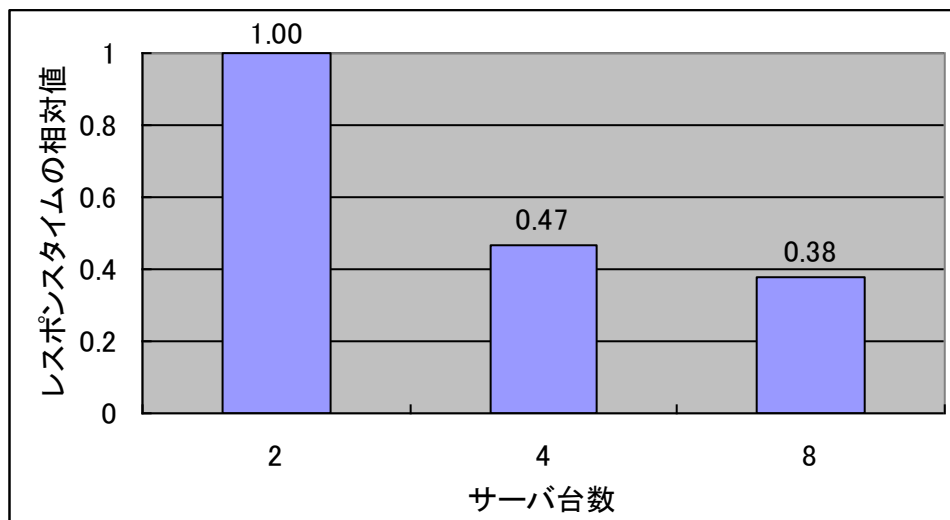


図 1-6 GET におけるクライアントが 100 の場合のサーバ台数増加に対する
処理時間の相対値の変化

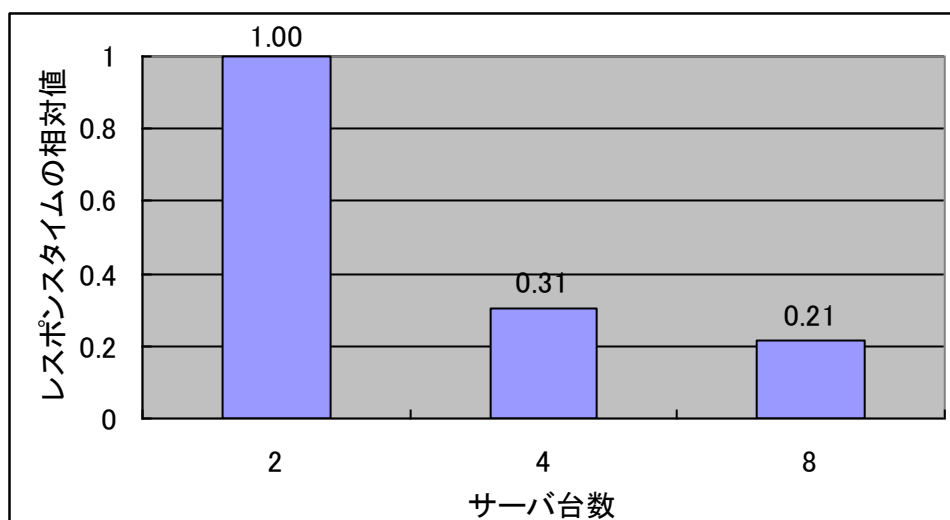


図 1-7 PUT におけるクライアントが 100 の場合のサーバ台数増加に対する
処理時間の相対値の変化

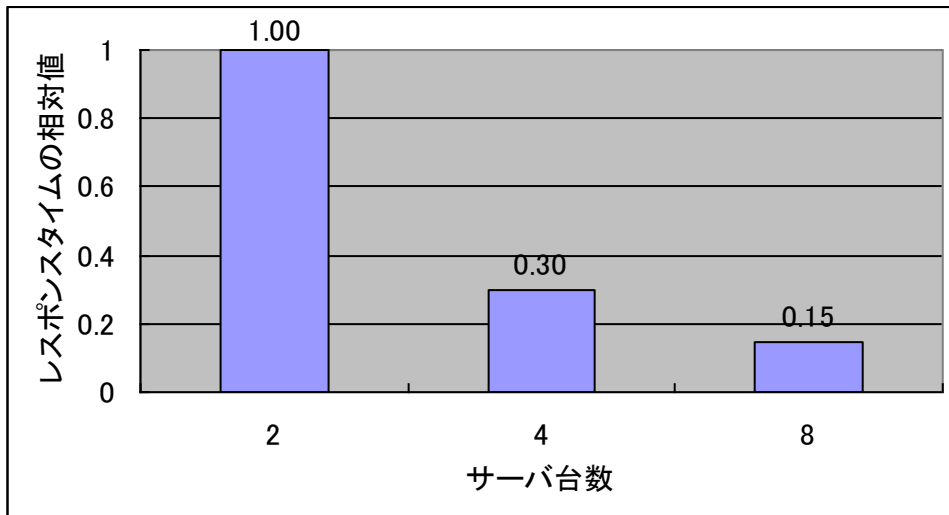


図 1-8 GET におけるクライアントが 1000 の場合のサーバ台数増加に対する
処理時間の相対値の変化

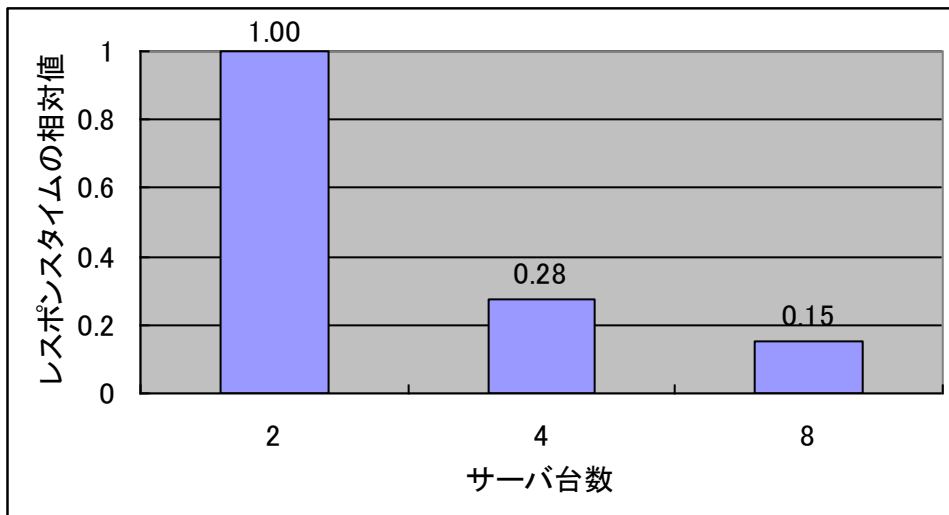


図 1-9 PUT におけるクライアントが 1000 の場合のサーバ台数増加に対する
処理時間の相対値の変化

図 1-6～図 1-9 共に、サーバ台数が増加すると、処理時間が短縮されることがわかる。その傾向は、サーバ台数に比例した傾向を示している。クライアント数が 100 の場合では、サーバ数が 4 台の場合の性能は、サーバ数が 2 台の性能の約倍程度、サーバ数が 8 台の場合の性能は、サーバ数が 4 台の性能の約 1.6~7 倍程度になっている。一方、クライアント数が 1000 の場合では、サーバ数が 4 台の場合の性能は、サーバ数が 2 台の約 3 倍程度まで伸びている。また、サーバ数が 8 台の場合の性能は、サーバ数が 4 台の約倍の性能になっている。この性能のばらつきに関してはネットワーク性能、サーバの CPU 性能、クライアントの CPU 性能などが考えられる。

1.2.3 評価シナリオ 2 : スループットからみたスケーラビリティ

(1) 計測方式

クライアントのアクセスは、評価シナリオ1と同様であるが、サーバ数を 2、4、8 のそれぞれの場合で、クライアント数を変化させ、スループット値が最大のを、それぞれのサーバ数での計測結果とする。クライアントのノード数は 20 である。

スループット値は下記の方法で計算する。

$$1000 \div (1000 \text{ 件の get/put に要した時間の平均}) \times (\text{クライアント数})$$

(2) 測定結果

図 1-10 と図 1-11 に GET、PUT それぞれのサーバ台数増加に伴うスループット向上の度合いを示す。

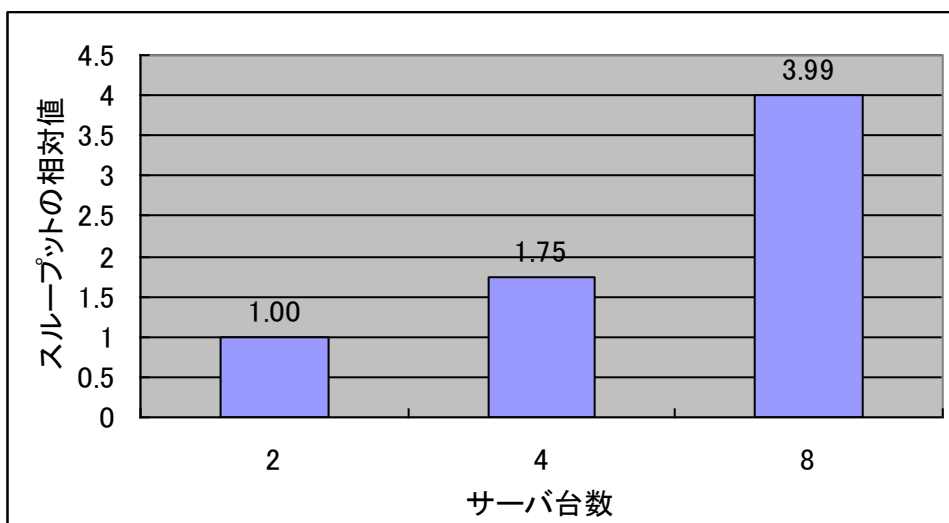


図 1-10 GET におけるサーバ台数増加に伴うスループットの向上

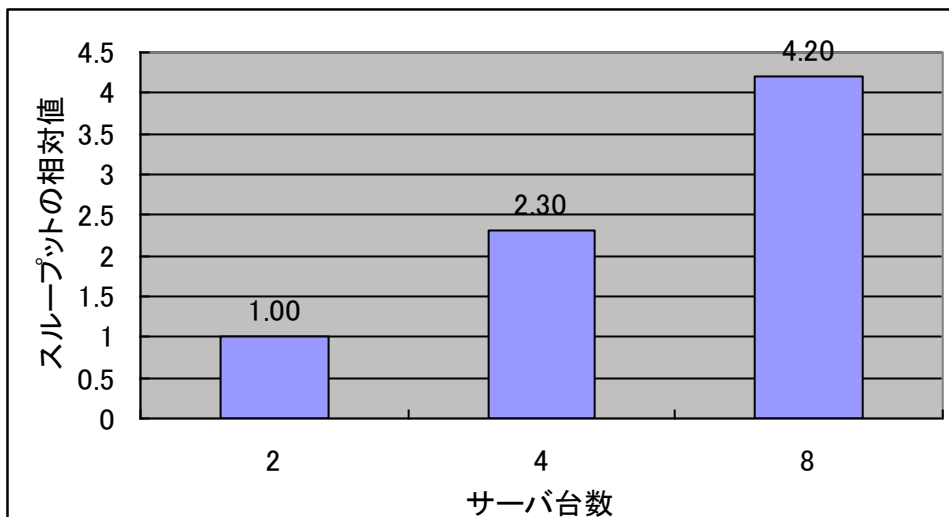


図 1-11 PUT におけるサーバ台数増加に伴うスループットの向上

計測時の多少のゆらぎはあるが、おおむね台数に比例してスループットが増加していることがわかる。

1.2.4 評価結果

本評価では、サーバ台数の増加に伴うレスポンスタイムの減少度合いとスループットの向上度合いを実測した。両方の実測値ともサーバ台数増加により性能が大幅に向上されることが確認された。

レスポンスタイムからみたスケーラビリティの評価では、サーバ台数が増加すると、クライアントのレスポンスタイムが短縮されることが確認された。ただし、レスポンスタイムは必ずしも1/サーバ台数にならないことも確認された。例えば、図 1-6 では、サーバ 8 台のレスポンスタイムは、4 台の場合のレスポンスタイムの半分になっていない。反対に、図 1-8、図 1-9 では、サーバ 4 台のレスポンスタイムは、2 台の場合のわずか 30%程度にまで低下しており、台数効果以上に短縮されている。

スループットからみたスケーラビリティの評価では、図 1-10、図 1-11 に見るように、台数に比例して性能が向上している結果が得られた。この評価では、システム全体のスループットが最大になるようにクライアント数を調整している。これらの計測では、サーバの CPU 使用率はおおよそ 100%近い状況になっている。このため、この計測では、サーバの性能を十分に引き出すことができたと言える。

1.2.5 考察

2つの評価結果ともに、WXS がサーバ台数に対して高いスケーラビリティを持つことが実証されたと言えるが、ここでさらに踏み込んで、2つの評価シナリオの結果に関して考察する。

(1) レスポンスタイムからみたスケーラビリティ評価の考察

レスポンスタイムからみたスケーラビリティの評価では、サーバ台数増加に伴い、クライアントのレスポンスタイムが短縮されることが確認されたが、レスポンスタイムの実測値が1/サーバ台数となっていない。実は、これは理論的にもそのようになるのである。レスポンスタイムは、ネットワークの処理時間やクライアント側の処理時間などサーバでの処理時間以外の部分の処理時間も大きく影響する。サーバ台数を増加させることは、この中のサーバでの処理時間を短縮させるだけである。この詳細について少し言及する。

例えば、図 1-6 では、サーバ台数が4台まででは、サーバのデータアクセスに伴う処理時間がレスポンスタイムに大きく影響を与えていると考えられる。そのため、サーバ台数 4 台のレスポンスタイムは 2 台のレスポンスタイムの約 50%程度になっている。一方、サーバ台数 8 台の場合では、すでにサーバでのデータアクセスの処理時間は短くなっており、全体のレスポンスタイムに対する影響は小さくなっている。実際、各サーバの CPU 利用率を見ると、サーバ台数 4 台の場合は 100%であったが、8 台の場合は、70~80%程度であった。結果として、8 台の場合のレスポンスタイムは 4 台の場合のレスポンスタイムの半分にはならなかったと考えられる。

図 1-8 では、別の現象を見ることができる。この評価では、クライアント数を 1000 としており、図 1-6、図 1-7 の評価以上にサーバへの負荷が高くなるため、クライアントのレスポンスタイム全体に対して、サーバでのデータアクセスの処理時間が大きく影響する。その結果、サーバ台数 8 台のレスポンスタイムはサーバ台数 4 台のレスポンスタイムの半分になっている。一方、サーバ台数 4 台のレスポンスタイムはサーバ台数 2 台のレスポンスタイムの 30%まで短縮されている。これは、サーバ台数 4 台の場合に非常に高い性能になったのではなく、サーバ台数 2 台の場合に、十分な性能を引き出せなかったということである。サーバ台数 2 台の場合では、サーバの CPU 利用率は 90%以下であった。また、スループットを計算すると、サーバ台数 2 台における一台あたりのスループット値は、サーバ台数 4 台の場合の半分程度であった。したがって、これはネットワークカードまた

はオペレーティングシステムの通信データ処理などのCPU以外のオーバーヘッド等が原因であると推測される。1000クライアントの要求メッセージが2台のサーバに集中した場合、ネットワークカードまたはオペレーティングシステムの通信データ処理が追いつかず、性能低下をもたらすであろうことは当然考えられる。これを裏付けるデータとして、サーバ台数が2台の場合のクライアント数が100の場合(図1-6のサーバ台数2の場合)のサーバ一台あたりのスループット値と1000の場合のスループット値(図1-8のサーバ台数2の場合)を比較すると、1000の場合のスループット値は100の場合の約半分程度の値となっていた。

このように、レスポンスタイムに関しては、システム内部の様々な部分での処理時間が関係しており、WXSサーバの部分だけ増強しても、その効果が十分引き出せない場合があることを十分に認識しておくことが大切である。

(2) スループットからみたスケーラビリティに関する考察

この評価結果では、サーバ台数に比例してスループット値が向上していることを示しており、非常にわかりやすい結果となっている。レスポンスタイムからみたスケーラビリティの評価では、クライアント数が定められている状況での性能計測であるため、サーバのCPU以外の部分がボトルネックとなることが発生した。一方、この評価では、できる限りサーバのCPUを使い切れるようにクライアント数を調整しながら計測した。そのため、多少の計測の揺らぎはあるものの、おおむねサーバ台数に比例したスループット値を得ることができたと言える。

ただし、単にサーバ台数を増やしていても、いずれはネットワークの性能がボトルネックとなり、性能向上を見込めなくなる限界点があることは認識しておく必要がある。

(3) 全体の考察

この評価結果全体を通して言えることは、データグリッドでは、サーバでのデータアクセス処理が非常に高速であるため、ネットワークがボトルネックになりやすいことを注意しておく必要がある、ということである。さらに、レスポンスタイムに関する考察で述べたように、ネットワーク全体の性能だけでなく、データグリッドサーバのネットワークカードやオペレーティングシステムの通信データ処理のオーバーヘッドも認識しておく必要がある。特に、サーバにアクセスするクライアント数が多い場合、ここがボトルネックとなる可能性がある。

複数のアプリケーションサーバと一台のデータベースサーバという伝統的なシステム構成では、多くの場合、データベースサーバがボトルネックとなった。データグリッドでは、スケールアウト構成によりサーバのボトルネックを解消する一方で、ネットワークなど別のボトルネックが生じる可能性がある。このことをよく勘案してシステム全体の構成を決める必要があることを、本評価の結果は示している。