Oracle GRID Center

HITACHI
Inspire the Next

ORACLE

WEB アプリケーションシステムにおける 性能拡張性と可用性の検証

Oracle Application Server 10g and Oracle Real Application Clusters 10g on 日立 BladeSymphony

Date: 2007 年 5 月
Version: 1.0

日本オラクル株式会社
システム製品統括本部 Grid Computing 技術部
中村智武
tomotake.nakamura@oracle.com

株式会社日立製作所
ソフトウェア事業部　オラクルビジネス統括センタ
矢田龍太郎
ryutaro.yada.jm@hitachi.com

1. はじめに

Web2.0 やロングテールといったキーワードが注目されているように、従来からあるオンラインショッピングや、ブログ、SNS などの新しいインターネット上のサービスが、多くの人々にとってますます身近になり、広く利用されている。このような WEB サービス利用者の爆発的な増加に対し、企業などのサービスプロバイダはそのシステムを増強し、利用者に快適なサービスを提供する必要がある。この際、新たにハイスペックなサーバーを用意して既存のサーバーをリプレースするのではなく、低コストサーバーを複数台並べてクラスタ構成にする、いわゆるスケールアウトと呼ばれる方法を採用するのが一般的になっていきている。スケールアウトはコストや可用性に優れ、特に WEB サーバー層、アプリケーション・サーバー層のシステム拡張に有効な性能拡張方法である。

2006 年 11 月、日本オラクル株式会社は、グリッドをベースとした次世代のビジネス・ソリューション構築を目的に、世界最大級規模の検証施設「Oracle GRID Center」を設立した。株式会社日立製作所はこれに賛同して、最新のサーバーおよびストレージを提供し、日本オラクル技術者と日立技術者による共同実機検証を推進している。Oracle の最新グリッド機能と日立の最新サーバーアーキテクチャを活かした最適構成や、仮想化の利用について検証し、国内外のお客様に成果を活用していただくことを最大の目的として取り組んでいる。

今回、日本オラクル株式会社と株式会社日立製作所は Oracle GRID Center にて、日立の高性能ブレード・サーバーBladeSymphony と Oracle Application Server 10g、Oracle Real Application Clusters 10g の組み合わせによる大規模な WEB アプリケーションベンチマーク検証を実施した。Oracle Application Server Cluster および Oracle Real Application Clusters 10g のそれぞれ最大 8 台のスケールアウト構成における性能およびそのスケーラビリティ、システムの可用性を実現するさまざまなレプリケーション方式の性能への影響を検証した結果を報告する。

謝辞

2006 年 11 月、日本オラクル株式会社は株式会社日立製作所やグリッド戦略パートナー各社と協業体制を確立し、企業のシステム基盤の最適化を実現する次世代のビジネス・ソリューションを構築するため、先鋭の技術を集結した「Oracle GRID Center(オラクル・グリッド・センター)」(http://www.oracle.co.jp/solutions/grid_center/index.html)を開設しました。本稿は、Oracle GRID Center の趣旨にご賛同頂いたインテル株式会社、シスコシステムズ株式会社のハードウェア・ソフトウェアのご提供および技術者によるご支援などの多大なるご協力を得て作成しております。ここに協賛企業各社およびご協力頂いた技術者に感謝の意を表します。

2. 目次

3. 検証目的

本検証の目的は、日立 BladeSymphony と Oracle Application Server 10g、Oracle Database 10g を組み合わせた WEB3 層システム構成において、J2EE WEB アプリケーションを使用した際のシステム性能とその性能スケーラビリティを検証することである。さらに、Oracle Application Server 10g が提供する WEB アプリケーションセッションの永続性を保証するレプリケーション機能を使用し、各方式の特徴とそれによる性能への影響を検証することで、システム設計におけるベストプラクティス見出すことである。

また、Oracle GRID Center における活動の大きな目的のひとつとして、検証で得られた成果やノウハウおよび事前検証済みのシステム構成を広く市場に展開し、SIer および顧客システム担当者に活用してもらうことで、Oracle のグリッド・インフラストラクチャを採用したシステム構築期間短縮とコスト削減の実現がある。今日の IT サービスは、オンライン・ショッピングや電子商取引などインターネットを介したサービスが主流であり、そのインフラはフロントに WEB サーバーやアプリケーション・サーバー、バックエンドにデータベース・サーバーといった構成が一般的であるが、このような WEB3 層システムの大規模な性能検証事例はあまり存在しなかった。従来のベンチマークでは、非現実的なストレージ構成や、通常運用では適用できないシステムチューニングを採用していることがあり、その性能データやその構成で得られるノウハウは現実的なシステムであまり活用できないケースが多い。我々の今回の検証では Oracle GRID Center のコンセプトにもとづき、現実的なシステム構成かつ汎用的なアプリケーションを採用している。

4. Oracle グリッド・インフラストラクチャ – Oracle Application Server Cluster 10g と Oracle Real Application Clusters 10g

Oracle のグリッド・インフラストラクチャは、クラスタリングされたアプリケーション・サーバーの Oracle Application Server Cluster と同じくクラスタリングされたデータベース Oracle Real Application Clusters 10g からなる。共に、プロセス監視や障害時のフェールオーバーなど、サービスを止めずに継続できる高い可用性と、スケールアウトによるリニアな性能拡張性を実現する。グリッド対応となった 10g からは、サービスに必要なリソースを業務ごとに適切かつ動的に割り当てることが可能になった。システム統合とグリッドによるリソース・マネジメントにより、サービスレベルとシステムコストを最適化することが可能となる。

今回の WEB アプリケーション性能検証は、Oracle Application Server Cluster と Oracle Real Application Clusters それぞれ 8 台によるシステム構成を採用し、レプリケーション機能による高可用性を実現しながら、ノードを追加した場合の性能拡張性を検証した。今回採用したアプリケーションはその特性上、データベースよりもアプリケーション・サーバーに対する負荷が高い。したがって、今回得られたシステム・スループットと性能拡張性はアプリケーション・サーバーのシステム拡張において非常に参考になるものである。データベースに関しては、今回と同等のトランザクションを使用したクライアント・サーバー型のアプリケーションによる性能検証を別途実施しているので、そちらの検証報告を参考にしていただきたい。

5. 日立 BladeSymphony BS320 について

今回の検証でアプリケーション・サーバーとして使用した日立の BladeSymphony BS320 は以下のような特徴をもち、WEB サーバー、アプリケーション・サーバー用途からミッドレンジ・システムにおけるデータベース・サーバーとして幅広く採用できるブレード・サーバーである。

超小型高集積でありながら最高性能を実現するブレード・サーバー

業界最高水準の省スペース(高さ 6U に 10 ブレード搭載)かつシャーシフル搭載時で約 98kg の軽重量、そして既存電源に適用可能な AC100V/200V に対応した導入が非常に容易なブレード・サーバーである。また、最新デュアル・プロセッサを搭載し、最高レベルの性能を実現する。

SAN ブート、N+1 スタンバイ機能による高信頼性の実現

SAN ブートおよび N+1 スタンバイ機能により、サーバー障害時に自動で障害ブレードを予備ブレードに切り替えることができ、これによりシステムの可用性を高め、万一のサーバー障害時にもサービスレベルを維持することが可能となる。

一元集約された統合運用管理

管理ソフトウェアを使用することで、ブレードのみならずストレージなど BladeSymphony 内すべてのシステム・リソースを、リモートからの統合運用管理することが可能である。システムのリソース監視だけでなく、ブレード・サーバーのデプロイメントやバックアップなども一元的に実施することが可能である。グリッド環境において非常に多くのサーバーを管理する際にこれらの機能は非常に重要である。本検証においても 16 台のブレード・サーバーをセットアップする際に、デプロイメント機能により、自動的に一括で OS 環境のセットアップを実施でき、システム構築コストを大幅に削減することができた。

6. Oracle Application Server 10g のレプリケーション機能

Oracle Application Server Cluster は、HTTP セッションまたはステートフル・セッション Enterprise JavaBean インスタンスに含まれるオブジェクトおよび値を OC4J インスタンス間でレプリケーションする機能を提供している(図 6-1 参照)。これにより、あるセッションで処理を実施中、そのセッションが接続しているアプリケーション・サーバーに障害が発生した場合に、レプリケーション先のアプリケーション・サーバーにフェールオーバーすることで、透過的にセッションを引き継ぐことが可能となる(図 6-2 参照)。

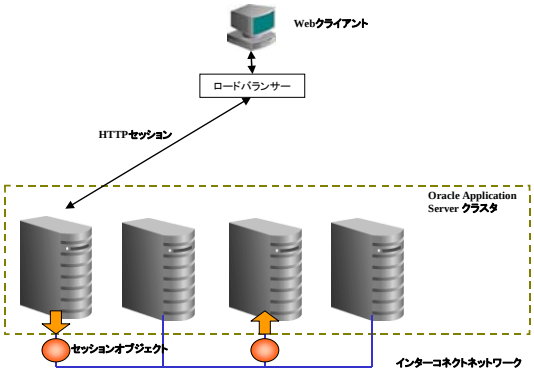


図 6-1
アプリケーションサーバクラスタ間のレプリケーション

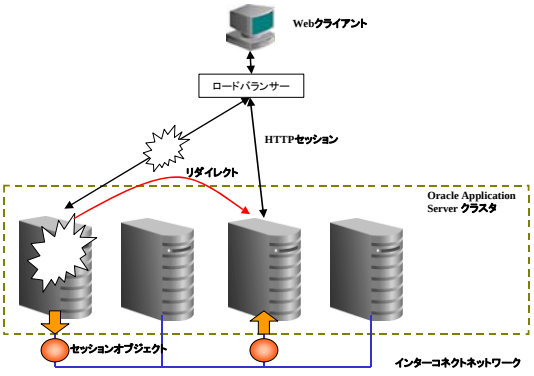


図 6-2
アプリケーションサーバクラスタ間のフェールオーバー

レプリケーション機能は転送方式や転送タイミングなどのポリシーを設定し、動作を制御することができる。本ホワイトペーパーでは、各レプリケーション設定時の挙動の特性とその性能への影響について検証している。
以下に設定可能なレプリケーションポリシーについて説明する。

転送方式(プロトコル)

- Peer-to-Peer
  TCP プロトコルを使用してクラスタ内の別の OC4J へセッションオブジェクトをレプリケーションする。レプリケーション先の OC4J の選択は、opmn プロセスが管理しているクラスタ情報を参照して決定する。

- Multicast
  UDP プロトコルを使用してクラスタ内の別の OC4J へセッションオブジェクトをレプリケーションする。各 OC4J はマルチキャストを使用して構成情報を他 OC4J に通知する。それぞれの OC4J は動的にクラスタ構成を認識し、レプリケーション先の OC4J を決定する。

- Database
  セッションオブジェクトを Oracle データベースへ格納する。

転送タイミング(トリガー)

- onSetAttribute
  値の変更時に、HTTP セッション属性に加えられた各変更をレプリケートする。プログラム的な面から見ると、HttpSession オブジェクトで setAttribute()がコールされるたびにレプリケーションが発生する。

- onRequestEnd(デフォルト)
  HTTP セッション属性に加えられたすべての変更をキューに入れ、HTTP レスポンスが送信される直前にすべての変更をレプリケートする。

- onShutdown
  JVM が正常に終了するたびに、HTTP セッションの現在の状態をレプリケートする。

転送グループ(スコープ)

- modifiedAttributes (デフォルト)
  変更された HTTP セッション属性、すなわち HttpSession オブジェクトで setAttribute()をコールして変更された値のみをレプリケートする。

- allAttributes
  HTTP セッションに設定されたすべての属性の値をレプリケートする。

同期方式

- 非同期
  データを他のインスタンスに非同期的にレプリケートする。

- 同期
  レプリケーションした際、レプリケーション先からのデータを受信したという確認応答を待機する。

Write-quota

レプリケーション先の数を指定する。write-quota のデフォルト値は 1 で、Oracle Application Server クラスタ内で自分以外の 1 つの OC4J にレプリケートする。Oracle

Application Server クラスタ内のすべての OC4J にセッションオブジェクトをレプリケートするには、クラスタ内の OC4J の合計数を write-quota の値として指定する必要がある。

7. 検証環境

7-1. システム構成

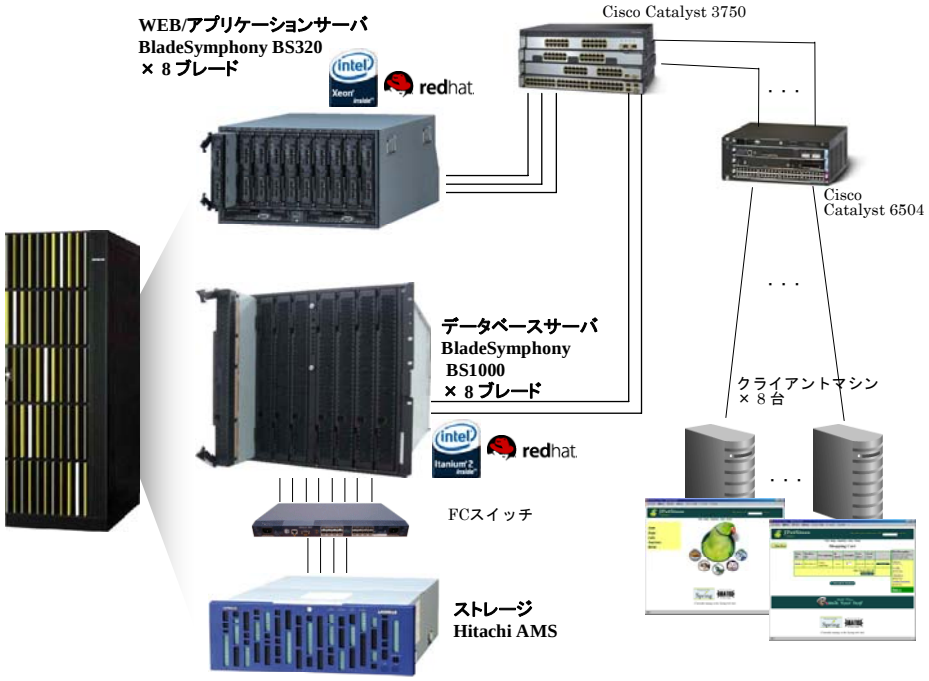


図 7-1 検証システム構成

7-2. 使用ハードウェア

- WEB/アプリケーション・サーバー

機種	日立 BladeSymphony BS320A × 8 ブレード
CPU	デュアルコア インテル®Xeon®プロセッサー 5160 3GHz 2 ソケット/ブレード
メモリ	8GB

- データベース・サーバー

機種	日立 BladeSymphony BS1000A × 8 ブレード
CPU	デュアルコア インテル®Itanium® 2 プロセッサー 9050 1.6GHz 2 ソケット/ブレード
メモリ	16GB

- クライアントマシン

機種	インテル開発検証用サーバー × 8 台
CPU	デュアルコア インテル®Xeon® プロセッサー 5150 2.66GHz 2 ソケット/サーバー

メモリ	4GB

- ストレージ

機種	Hitachi AMS
ハードディスク	144GB × 28HDD (+ 2HDD スペア)
RAID グループ構成	1D+1P × 2 (OS 用) 2D+1P × 8 (Oracle データベース用)

- ネットワーク・スイッチ

機種	Catalyst 6504 Catalyst 3750

7-3. 使用ソフトウェア

- WEB/アプリケーション・サーバー

OS	Red Hat Enterprise Linux ES 4 Update 1
Oracle	Oracle Application Server 10g 10.1.3.1

- データベース・サーバー

OS	Red Hat Enterprise Linux AS 4 Update 4
Oracle	Oracle Database 10g Enterprise Edition 10.2.0.3 Oracle Real Application Clusters 10g Oracle Partitioning

- クライアントマシン

OS	Red Hat Enterprise Linux AS 4 Update 3
Oracle	Oracle Client 10.2.0.1
負荷生成ツール	Apache JMeter 2.2

7-4. ネットワーク構成

図 7-2にネットワーク構成の詳細図を示す。BladeSymphonyの各ブレードにはネットワーク・インターフェースが 4 つあり、それらはすべてシャーシに内蔵されている 2 つのネットワーク・スイッチに内部的に接続されている。今回はVLAN機能により、eth0 はパブリック・ネットワークとして内蔵スイッチ 1 を使用し、eth1 はレプリケーション転送用のインターコネクトネットワークとして内蔵スイッチ 2 を使用し設定をした。eth2 とeth3 は未使用である。内蔵スイッチとCiscoのスイッチの間はギガビット・イーサケーブルで接続してある。

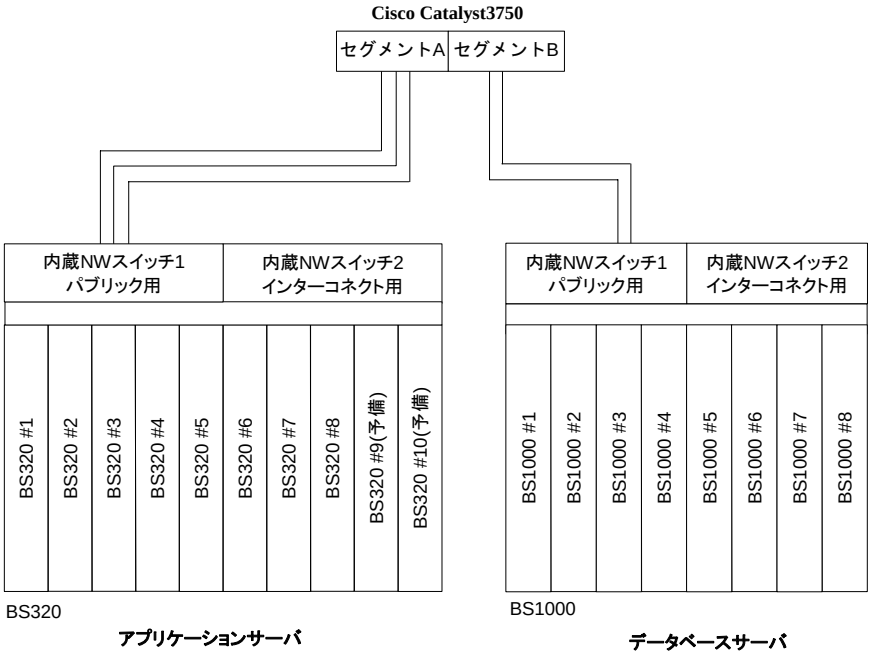


図 7-2　ネットワーク構成

7-5. ベンチマークアプリケーション

本検証では、ベンチマーク用のアプリケーションとして JPetStore(注１)を使用した。JPetStore は、Spring Framework のサンプル・アプリケーションとして作成されたもので、Web ショッピングサイトを想定して設計されている。JPetStore では、ログイン、商品を検索、任意の商品を商品カートに追加、商品を購入(トランザクションのコミット)、ログアウトといった一般的な OLTP 処理を実行することができ、アプリケーション・サーバー、データベース・サーバーに効率よく負荷をかけることができる。また、アプリケーション上で生成されるオブジェクトは HttpSession に保持されるため、セッション・レプリケーションの検証を行う上でも最適なアプリケーションである。

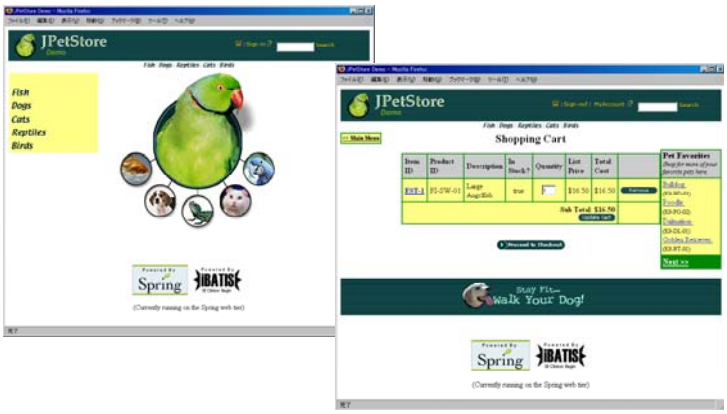


図 3　JPetStore アプリケーション

7-6. Apache JMeter の設定

Oracle Application Server に配布した JPetStore に対して、クライアントから負荷をかけるツールとして Apache JMeter を利用した。Apache JMeter を利用することで、ログインからログアウトするまでの一連の処理をシナリオ化して実行させることができる。さらに、同時実行スレッド数、待機時間を設定することにより、アプリケーション・サーバーに対する負荷量を調節することができる。

JPetStore シナリオ

本検証では、以下のシナリオを Apache JMeter で定義し、全ての計測で使用した。

1. ログイン
   ランダムで任意のユーザー名を決定し、そのユーザー名とパスワードを使用してログインを行う。
   OC4J 上では、ユーザー情報が HttpSession に保持される。

2. 商品を検索
   ランダムで検索キーワードを決定し、商品を検索する。検索結果は平均１００件になるように調整している。
   OC4J 上では、ページングに使うための検索結果が HttpSession に保持される。

3. 商品を選択
   検索された商品一覧からランダムで一つのアイテムを選択する。
   OC4J 上では、アイテム情報が HttpSession に保持される。

4. 選択した商品を商品カートに追加
   OC4J 上では、商品カートが HttpSession に保持される。

5. 繰り返し
   「２．商品を検索」から「４．選択した商品を商品カートに追加」までを１回から５回までのランダム回数実行する。

6. 購入
   商品カートに入っているアイテムを購入する。
   データベースに対してトランザクションが実行される。

7. ログアウト
   OC4J 上では、全ての HttpSession を無効化する。

Apache JMeter パラメータ

Apache JMeter 上では、以下のパラメータを設定している。クライアントから生成される負荷量に関係する。

同時実行スレッド数	350, 700	クライアントから負荷をかけるために起動するスレッド数。多重度。
待機時間	300ms±100ms	ページ遷移をするたびに待機する時間。指定範囲内でランダムな値を選択している。

7-7. Oracle HTTP Server のロードバランシング設定

Oracle HTTP Server(OHS)から OC4J へのロードバランシング機能はポリシーを設定することでその動作を制御できる。設定可能なポリシーは以下の 8 つである。

- Random
- Round Robin
- Random with Local Affinity
- Round Robin with Local Affinity
- Random using Routing Weight
- Round Robin using Routing Weight
- Metric Based
- Metric Based with Local Affinity

本検証では、OHS のロードバランシングポリシーとして Round Robin with Local Affinity を設定している。この設定は、OHS が実行されているホスト上で OC4J が起動していた場合には、その OC4J に優先してルーティングする設定である。本構成では、各ノード上で OHS と OC4J の両方を起動しているため、OHS・OC4J 間の通信にネットワークが使用されない。そのため、ネットワークの帯域を有効に活用できる他、クライアントからアプリケーション・サーバー間のネットワーク・トラフィック量、もしくは OC4J セッション・レプリケーションに使用されるネットワーク・トラフィック量を正確に測定できる利点がある。

8. 検証内容

以下の項目について測定を実施した。

8-1. 1 ノードシステムのスループット-レスポンスタイムの測定

アプリケーション・サーバー、データベース・サーバーそれぞれ N ノードで構成されるシステムを N ノードシステムと定義する。1 ノードシステムにおいて、アプリケーションの最大スループットを測定する。また、8 ノードシステムまでの性能スケーラビリティを検証するにあたり、1 ノードシステムにおいて、最適なスループットとレスポンスタイムを維持する限界のアプリケーション負荷量を検証する。レプリケーションの設定は行わない。

8-2. 最大 8 ノードシステムにおけるスループットとスケーラビリティ

2 ノードから最大 8 ノードシステムでのスループットとレスポンスタイムを測定し、スケールアウトによるシステム拡張時の性能スケーラビリティを検証する。1 ノードあたりのアプリケーション負荷量は、1 ノードシステムにおいて最適なスループットとレスポンスタイムを維持する限界の負荷量とする。レプリケーションの設定は行わない。

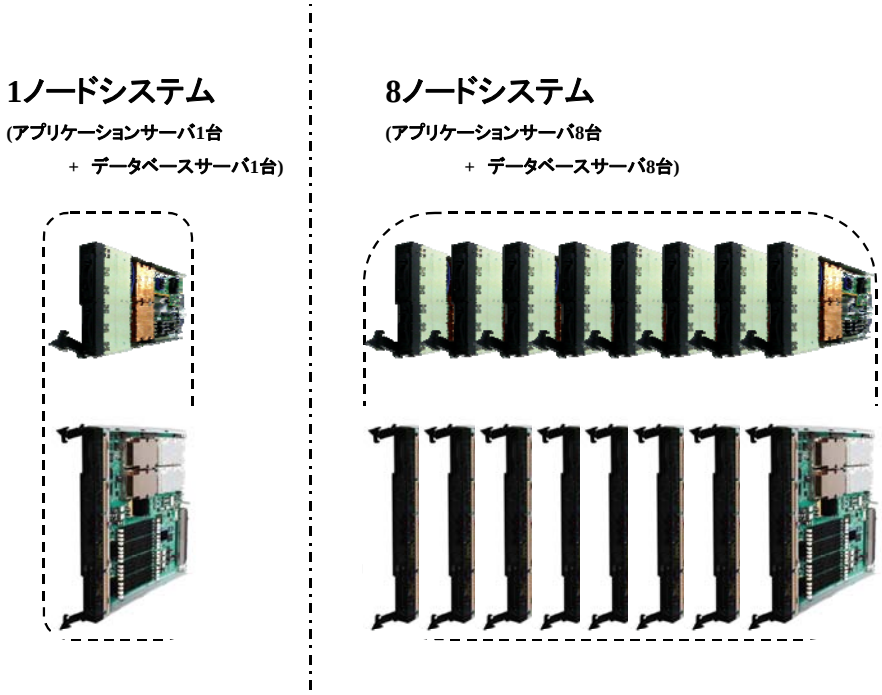


図 8-1　N ノードシステムの定義

8-3. セッション・レプリケーション設定によるスループットと挙動の違い

セッション・レプリケーションを有効化した設定において、2 ノードから最大 8 ノードシステムでのスループットとレスポンスタイムを測定する。クライアントからの負荷が高い場合と中程度の負荷である場合のそれぞれで測定する。高負荷では、セッション・レプリケーションによるスループットへの影響およびスケーラビリティへの影響を、中負荷ではセッション・レプリケーションによる CPU 使用率やネットワーク・トラフィックなどシステム・リソースの負荷の違いを考察する。

また本ホワイトペーパでは、セッション・レプリケーション設定について、以下の項番をもとに次のように表現する。

パターン ABCD
- A: 転送方式(プロトコル)
- B: 転送タイミング(トリガー)
- C: 転送グループ(スコープ)
- D: 同期方式

A、B、C、D にはそれぞれ次の値を取る。

A: 転送方式(プロトコル)
1. Multicast
2. Peer-to-Peer
3. Database

B: 転送タイミング(トリガー)
1. onSetAttribute

2. onRequestEnd
3. onShutdown

C: 転送グループ(スコープ)
1. modifiedAttributes
2. allAttributes

D: 同期方式
1. 非同期
2. 同期

例：
パターン 2211 (Peer-to-Peer / onRequestEnd / modifiedAttributes / 非同期 を意味する)
パターン 2112 (Peer-to-Peer / onSetAttribute / modifiedAttributes / 同期 を意味する)

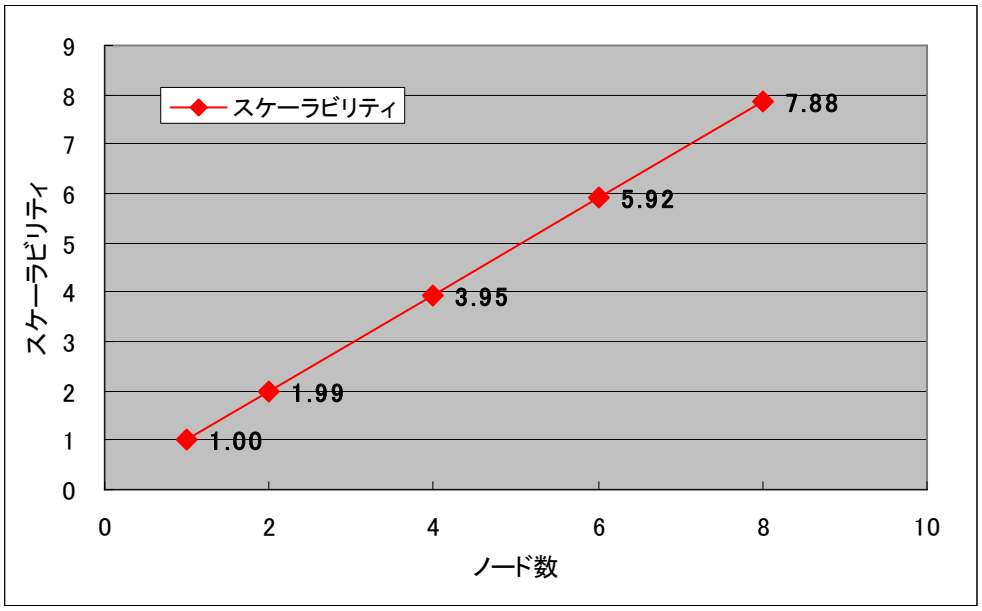
以下、表 8-1が今回測定したレプリケーションパターンの一覧である。

測定パターン	転送方式(プロトコル)	転送タイミング(トリガー)	転送グループ(スコープ)	同期方式	Write quota
パターン 2211	Peer-to-Peer	onRequestEnd	modifiedAttributes	非同期	1
パターン 2212	Peer-to-Peer	onRequestEnd	modifiedAttributes	同期	1
パターン 2111	Peer-to-Peer	onSetAttribute	modifiedAttributes	非同期	1
パターン 2112	Peer-to-Peer	onSetAttribute	modifiedAttributes	同期	1
パターン 2121	Peer-to-Peer	onSetAttribute	allAttributes	非同期	1
パターン 2122	Peer-to-Peer	onSetAttribute	allAttributes	同期	1
パターン 2221	Peer-to-Peer	onRequestEnd	allAttributes	非同期	1
パターン 2222	Peer-to-Peer	onRequestEnd	allAttributes	同期	1
パターン 1211	Multicast	onRequestEnd	modifiedAttributes	非同期	1
パターン 2311	Peer-to-Peer	onShutdown	modifiedAttributes	非同期	1
パターン 2211wq2	Peer-to-Peer	onRequestEnd	modifiedAttributes	非同期	2
パターン 2211wq3	Peer-to-Peer	onRequestEnd	modifiedAttributes	非同期	3

表 8-1　測定パターン一覧表

9. 検証結果要約

次のグラフは、Oracle Application Server 10g をクラスタ構成で 1 ノードから最大 8 ノードまで拡張したときの性能検証結果である。

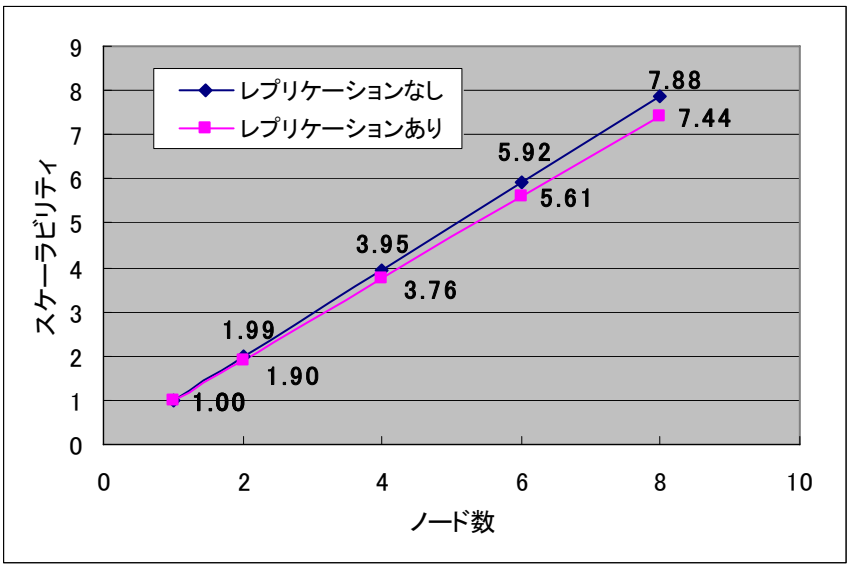


グラフ 9-1 Oracle Application Server Cluster スケーラビリティ検証結果

グラフ 9-1 は、Oracle Application Server 10g を 1 ノードから 8 ノードまでクラスタリング構成を拡張した際のスループットを比較している。1 ノードでのスループットを 1.00 としたときの 2 ノードでのスループット比は 1.99、4 ノードでは 3.95、8 ノードでは 7.88 となっており、拡張されるノード数に比例して性能が向上していることがわかる。

これは、Oracle Application Server Cluster の高い拡張性を示している。負荷に応じて必要なリソースを割り当てられることを意味し、Oracle GRID の基盤として Oracle Application Server 10g が最適なコンポーネントであると言える。

次に、Oracle Application Server 10g の J2EE 実行環境である、OC4J のセッション・レプリケーション機能の性能について検証した結果について報告する。セッション・レプリケーションとは、ある OC4J 上で HttpSession に保持された情報を他の OC4J にレプリケートする機能である。これによりあるアプリケーション・サーバーに障害が発生した際に HTTP セッションを透過的にフェイルオーバーすることができ、アプリケーション・サーバー層において高い可用性を実現することができる。



グラフ 9-2 セッション・レプリケーション設定によるスループット比較

グラフ 9-2 は、Oracle Application Server 10g を 1 ノードから 8 ノードまで拡張した際の OC4J セッション・レプリケーションの有無によるスループットを比較している。

セッション・レプリケーションの設定を行うと、HttpSession が保持するオブジェクトを他ノードに伝播するため、オーバーヘッドが生じる。本検証では、セッション・レプリケーションによるスループットの低下は 5%程度であった。8 ノード構成までにおいて、ノードの拡張数に比例してスループットも増加しており、セッション・レプリケーションの設定を行っていても高いスケーラビリティを実現できている。これは、J2EE アプリケーション層において高い可用性を保ちつつ、拡張性も実現できることを示している。

以上の結果より、Oracle Application Server 10g ではクラスリングを行うことにより拡張性と可用性の両立を実現できることが確認された。

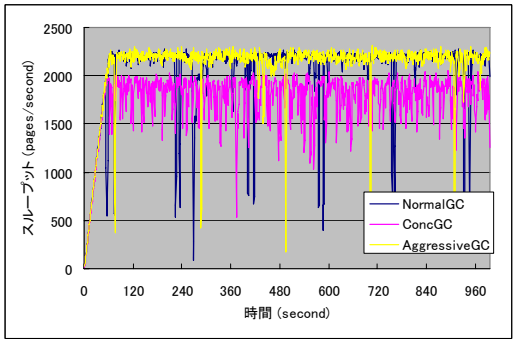
次の章から、検証結果の詳細な内容について報告する。

10. 検証結果詳細

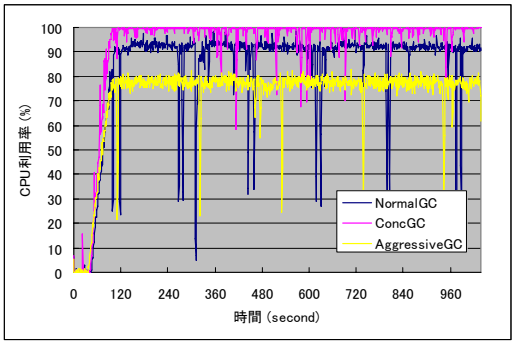
10-1. Oracle Application Server 10g 基礎性能検証

Oracle Application Server 10g が 1 ノードのみのシングル構成での性能検証である。この検証をスケーラビリティ検証の基礎検証として実施した。シングル構成で JVM パラメータおよびクライアントからの負荷量を調整し、1 ノードでの最適な条件を決定する。ここで決定された条件をクラスタ構成でも適用することにより、ノード構成を変更した際の性能を比較している。

10-1-1. JVM ガベージ・コレクションの設定による挙動の変化



グラフ 10-1
JVM 起動オプションによるスループット推移の違い



グラフ 10-2
JVM 起動オプションによる CPU 使用率の違い

グラフ 10-1 とグラフ 10-2 は、それぞれ 1 ノードのみのアプリケーション・サーバーに対して負荷をかけたときの、JVM のガベージ・コレクション（以下 GC）方式の違いによるスループットと CPU 使用率の違いである。この検証は、スケーラビリティ検証におけるパラメータを決定するための基礎検証として実施した。比較しているのは、以下のガベージ・コレクション方式である。

方式	Java 起動オプション	概要
通常の GC	なし	デフォルトで選択されている GC 方式。GC はシングル処理で実行される。
コンカレント GC	-XX:+UseConcMarkSweepGC	多くの GC がアプリケーション実行を止めることなく行われる。
アグレッシブ・ヒープ	-XX:AggressiveHeap	ヒープサイズが自動で調整される。GC がパラレル処理で実行される。

通常の GC およびコンカレント GC では最大メモリサイズを 2GB に設定している。アグレッシブ・ヒープでは最大メモリサイズを設定していない。今回の環境は 32bit OS なので、アグレッシブ・ヒープでは 2GB の範囲内でヒープサイズが自動調整される。

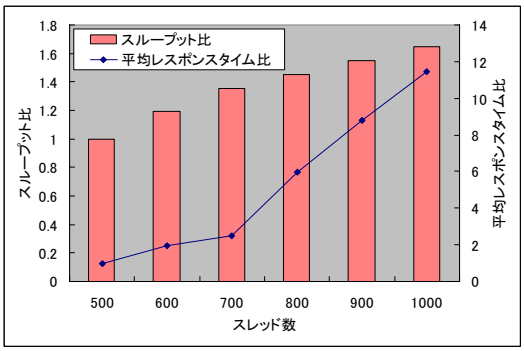
今回の検証環境においては、アグレッシブ・ヒープ方式が非常に効果があった。アプリケーション・サーバーの CPU 使用率が通常の GC およびコンカレント GC に比べると

10~15%程度低く抑えられて同じ程度のスループットが出ていた。また、グラフ 10-1 および 10-2 を見ると定期的な値の落ち込みが発生しているが、これはフル GC が発生しているためにアプリケーションが瞬間的に止まっていることを示している。この落ち込みの発生頻度を見ても、アグレッシブ・ヒープ方式では比較的緩やかで、アプリケーションをより安定的に実行できることが確認できた。

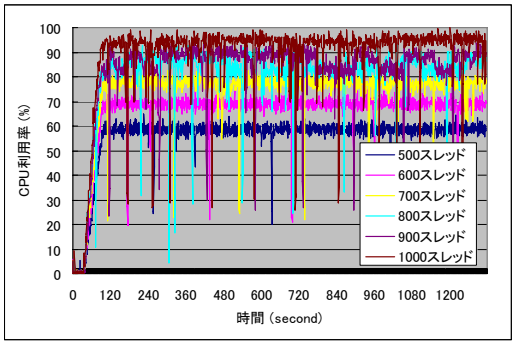
以上の結果から、アグレッシブ・ヒープ方式を最適なパラメータとして以降の検証では設定している。

10-1-2. 負荷量の違いによるスループットとレスポンスタイムの推移

グラフ 10-3 とグラフ 10-4 は、それぞれ、1 ノードシステムにおいてクライアントからの同時実行スレッド数を 500,600,700,800,900,1000 と増加させていった際のスループットとレスポンスタイム、およびアプリケーション・サーバーの CPU 使用率の測定結果である。負荷量を増加させていくと CPU の使用率と共にスループットは単調に増加し、負荷量 1000 スレッドで CPU 使用率が 100%となった。レスポンスタイムは、負荷量が 700 スレッドを超えたあたりから急激に増加しているが、これはセッション数増大によるシステムオーバヘッドが顕著になっているためと考えられる。レスポンス・タイムが 700 スレッドまで安定して推移していることから、700 スレッドが 1 ノードシステムにおいて最適にサービスを提供できる限界の負荷量であると判断できる。なお、データベース・サーバーに関しては、負荷量が一番多い 1000 スレッドにおいても CPU 使用率は 40%程度であり、ボトルネックとはなっていないことがわかる。



グラフ 10-3
1 ノードシステムにおけるスループットとレスポンスタイム



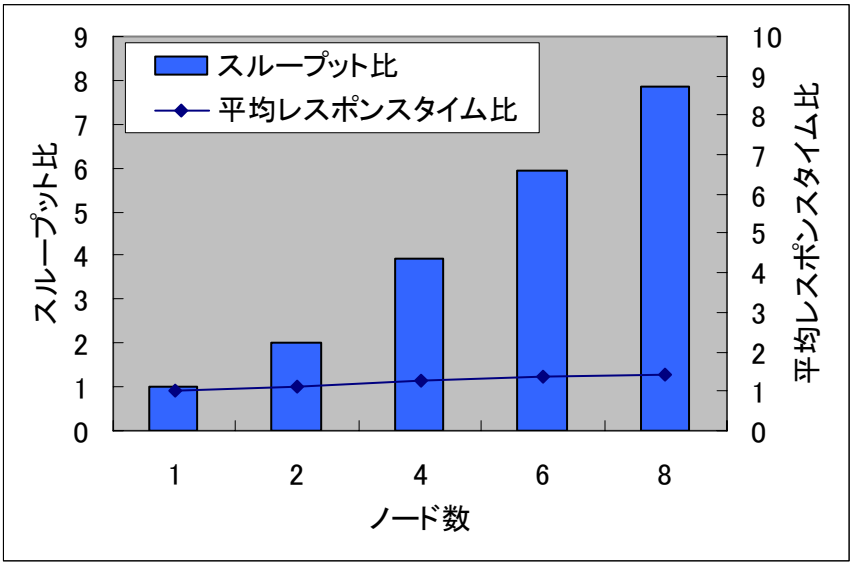
グラフ 10-4
1 ノードシステムにおける CPU 使用率

以降の検証では、同時実行スレッド数 700 とその半数の負荷量である同時実行スレッド数 350 を使用して性能を測定している。

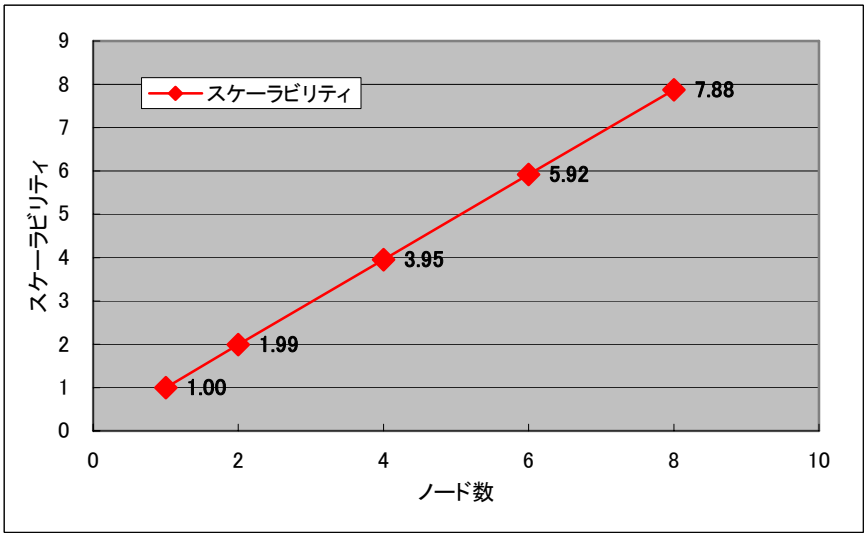
10-2. Oracle Application Server 10g スケーラビリティ

グラフ 10-5 は 1 ノードから 8 ノードまでアプリケーション・サーバーとデータベース・サーバーのノード数を拡張した際のスループットとレスポンスタイムを比較したものである。アプリケーションの負荷量は 1 ノードあたり 700 スレッドである。スループットはノード数にほぼ比例して増加しており、レスポンスタイムはノード数に関係なく安定している。この検証では、OC4J のセッション・レプリケーションを使用していない。この結果から、レプリケーション無しの Oracle AS クラスタ構成では、ノード数が増えてもクラスタ構成によるオーバーヘッドがほとんどなく、スケールアウトによりシステムの性能を拡張できることが分かる。

グラフ 10-6 は、1 ノードシステムでのスループットを 1.00 とした場合のそれぞれのノード構成におけるスループット比を表している。2 ノードでは 1.99、4 ノードでは 3.95、8 ノードで 7.88 となっており、非常に性能拡張性が高いことが分かる。



グラフ 10-5 スループットとレスポンスタイム比較



グラフ 10-6　性能スケーラビリティ

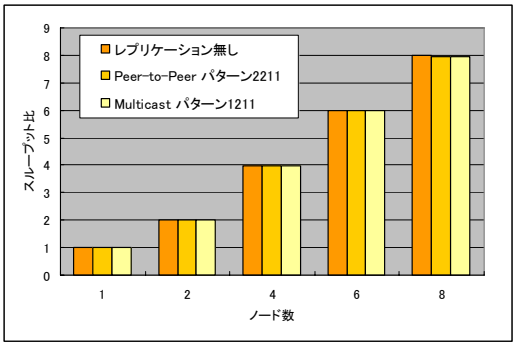
10-3.　レプリケーション方式によるスループットと挙動の違い

以降は、2 ノードから最大 8 ノードまでのクラスタ構成において、OC4J セッション・レプリケーションを有効にした際の性能検証について報告する。クライアントからの負荷が中負荷である場合と高負荷である場合について性能を測定し、セッション・レプリケーションの設定を行っていない場合の性能と比較することによりその特性を把握する。

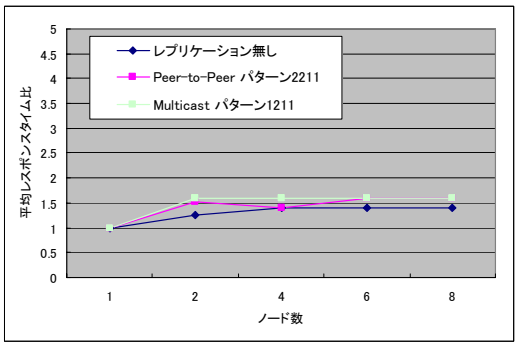
10-3-1. 転送方式(プロトコル)による挙動の違い

中負荷

グラフ 10-7 とグラフ 10-8 は、レプリケーション無し、Peer-to-Peer 、Multicast それぞれのレプリケーション転送方式での、中負荷におけるスループットとレスポンタイムを比較したものである。CPU リソースに余裕がある中負荷においては、各転送方式でのスループットの差はほとんどない。またレスポンスタイムも同様に差がなく、Peer-to-Peer および Multicast 転送方式によるレプリケーションを設定していても、アプリケーションの応答時間には影響がないことがわかった。



グラフ 10-7
転送方式(プロトコル)によるスループットの違い(中負荷 350 スレッド)

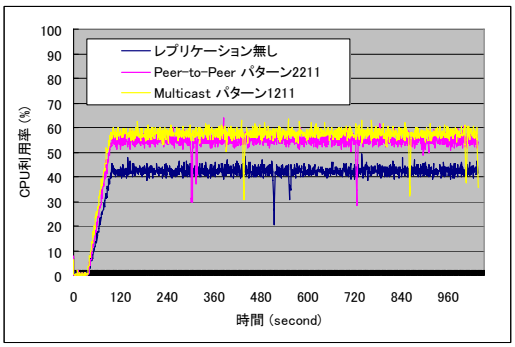


グラフ 10-8
転送方式(プロトコル)によるレスポンスタイムの違い(中負荷 350 スレッド)

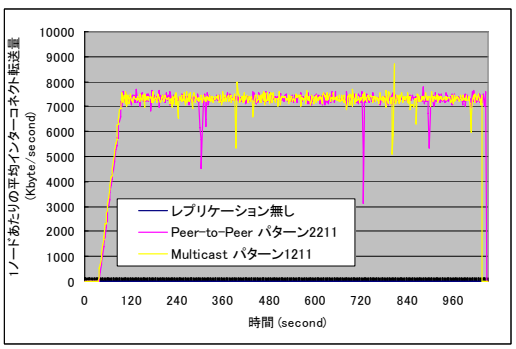
次に、各転送方式における CPU 使用率をグラフ 10-9 で、インターコネクト転送量の違いをグラフ 10-10 で示している Peer-to-Peer もしくは Multicast のレプリケーション方式を有効にすることで、15%程度の CPU 使用率の増加が確認できる。これは各ノードが他ノードに対しセッション・オブジェクトをレプリケートする処理のオーバーヘッドであると考えられる。

レプリケートされるセッションオブジェクトの転送量は、Peer-to-Peer と Multicast の転送方式で差がない。ノード数が 2 ノードから 8 ノードまで増加しても、ノードあたりの平均インターコネクト転送量は一定であった。これは、どちらの転送方式もセッション・オブジェクトのレプリケーション先は総ノード数に関係なくある一つのノードであるからである。

これら結果から、中負荷において Peer-to-Peer と Multicast の転送方式によってアプリケーションの性能への影響に違いはないといえる。



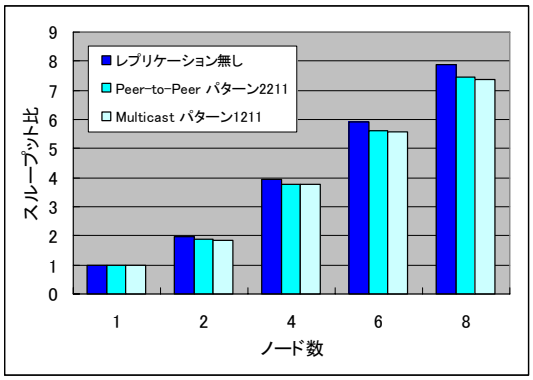
グラフ 10-9
転送方式(プロトコル)による CPU 使用率の違い(中負荷 350 スレッド)



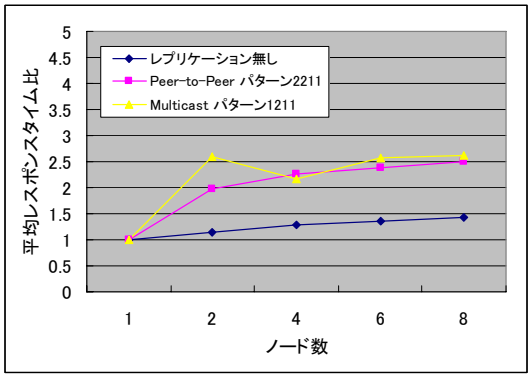
グラフ 10-10
転送方式(プロトコル)によるインターコネクト転送量の違い(中負荷 350 スレッド)

高負荷

グラフ 10-11 とグラフ 10-12 は、レプリケーション無し、Peer-to-Peer、Multicast それぞれのレプリケーション転送方式での、高負荷におけるスループットとレスポンタイムを比較したものである。CPU リソースに余裕がない高負荷の条件では、オーバーヘッドがスループットとレスポンスタイムの低下として現れた。Peer-to-Peer および Multicast では、レプリケーションを行わない場合に比べて 5%程度スループットが低下した。レスポンスタイムは Peer-to-Peer、Multicast 共に約 2 倍程度伸びた。

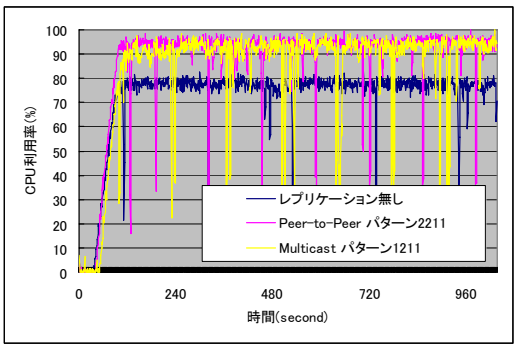


グラフ 10-11
転送方式(プロトコル)によるスループットの違い(高負荷 700 スレッド)

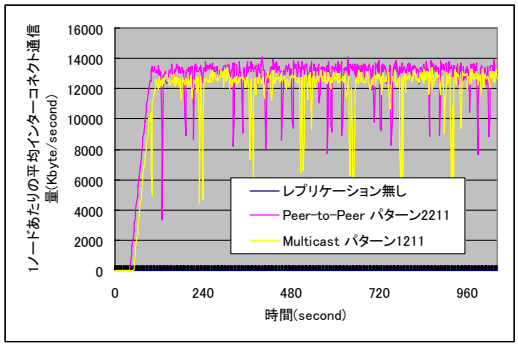


グラフ 10-12
転送方式(プロトコル)によるレスポンスタイムの違い(高負荷 700 スレッド)

次に、CPU 使用率とインターコネクト転送量の違いを示す。こちらは、中負荷の条件と同様 Peer-to-Peer と Multicast にはほぼ違いがなかった。



グラフ 10-13
転送方式(プロトコル)による CPU 使用率の違い(高負荷 700 スレッド)



グラフ 10-14
転送方式(プロトコル)によるインターコネクト転送量の違い(高負荷 700 スレッド)

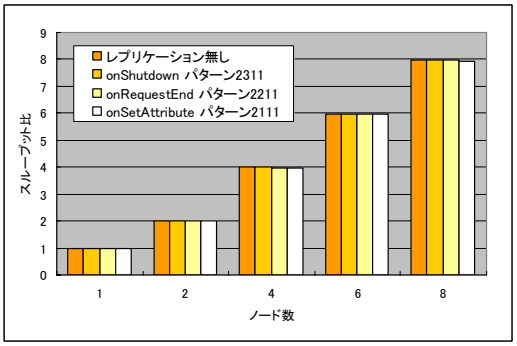
10-3-2. 転送タイミング(トリガー)による挙動の違い

中負荷

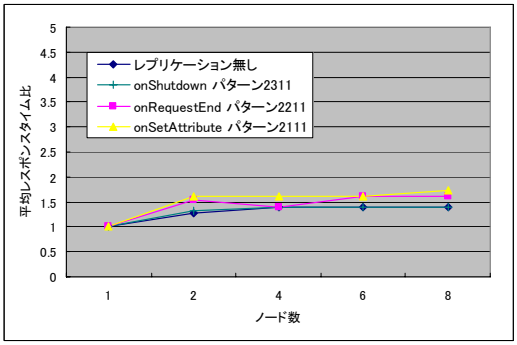
グラフ 10-15 とグラフ 10-16 は、レプリケーション転送タイミング(トリガー)を onShutdown、onRequestEnd、onSetAttribute と設定した場合の、中負荷におけるスループットとレスポンスタイムの測定結果である。各転送タイミングによってスループットとレスポンスタイムに大きな差は無かった。

グラフ 10-17 はそれぞれの設定におけるインターコネクト転送量の違いを示している。onSetAttribute がもっとも転送量が多く 9Mbyte/秒程度である。onRequestEnd は onSetAttribute より約 2Mbyte/秒少ない。このネットワーク処理量の差によって、onSetAttribute の方が onRequestEnd より CPU 使用率が高くなっていることがグラフ 10-18 から分かる。クライアントからの一つの HTTP リクエストで複数のセッション・オブジェクトを設定した場合、onRequestEnd では一回でレプリケーションが行われるが、onSetAttribute では setAttribute()がコールされるごとにそれぞれのセッション・オブジェクトがレプリケートされるためオーバーヘッドが大きくなる。onSetAttribute のオーバーヘッドは 1 リクエストでの setAttribute()コール数に依存するため、性能特性はアプリケーションにより異なる。

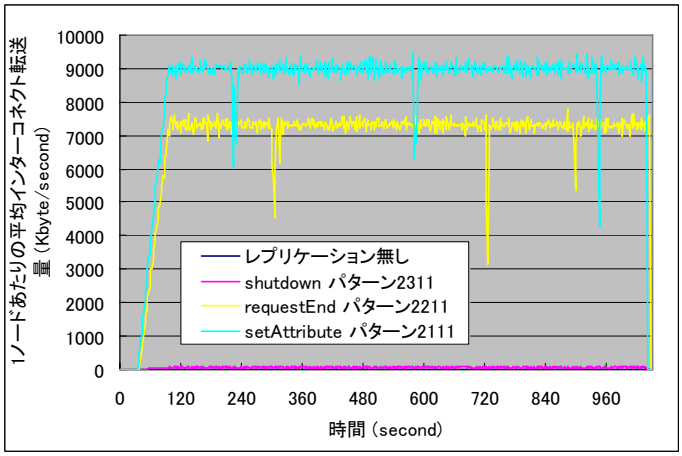
onShutdown は、レプリケーションなしの場合とほとんど違いがないが、約 70Kbyte/秒のインターコネクト転送量があった。これはレプリケーションによるものではなく、クラスタ間の制御情報の通信によるものである。



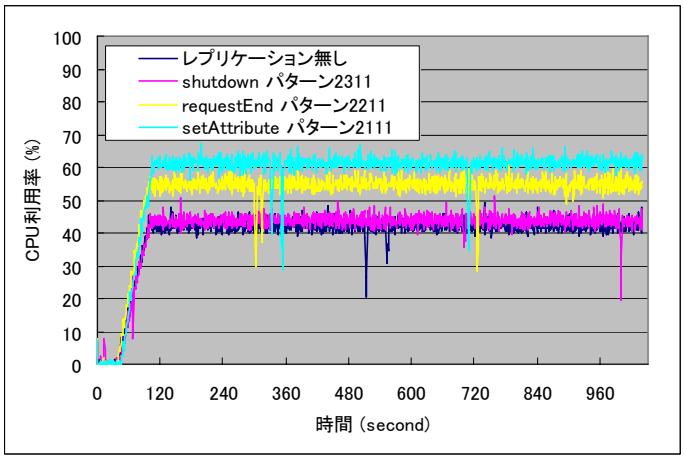
グラフ 10-15
転送タイミング(トリガー)によるスループットの違い(中負荷 350 スレッド)



グラフ 10-16
転送タイミング(トリガー)によるレスポンスタイムの違い(中負荷 350 スレッド)



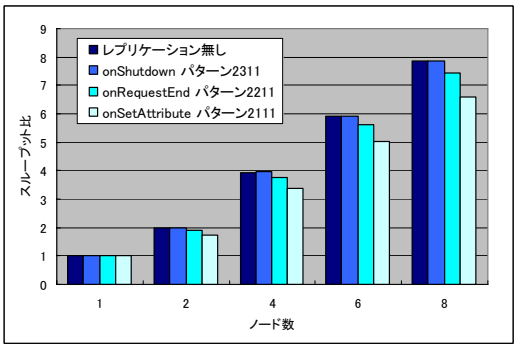
グラフ 10-17 転送タイミング(トリガー)によるインターコネクト転送量の違い(中負荷 350 スレッド)



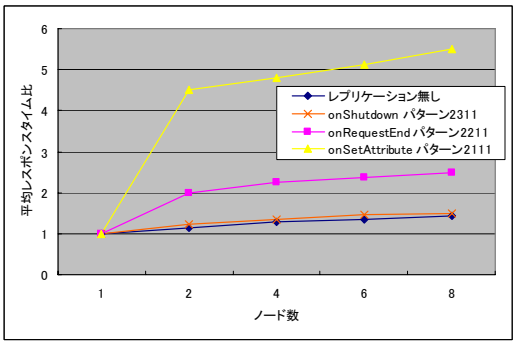
グラフ 10-18　転送タイミング(トリガー)による CPU 使用率の違い(中負荷 350 スレッド)

高負荷

グラフ 10-19 およびグラフ 10-20 は、レプリケーション転送タイミング(トリガー)を onShutdown、onRequestEnd、onSetAttribute と設定した場合の、高負荷におけるスループットとレスポンスタイムの測定結果である。onShutdown はレプリケーションなしとほとんど違いがないが、onRequestEnd および onSetAttribute では 8 ノード構成において、それぞれ 6%, 16%程度の性能劣化が見られる。onSetAttribute の性能劣化が大きいのは、中負荷の結果と同様に onSetAttribute の方がネットワーク転送量が多く、CPU リソースを必要とするためである。このオーバーヘッドにより onSetAttribute ではレスポンスタイムが 3～4 倍増加している。



グラフ 10-19
転送タイミング(トリガー)によるスループットの違い(中負荷 700 スレッド)

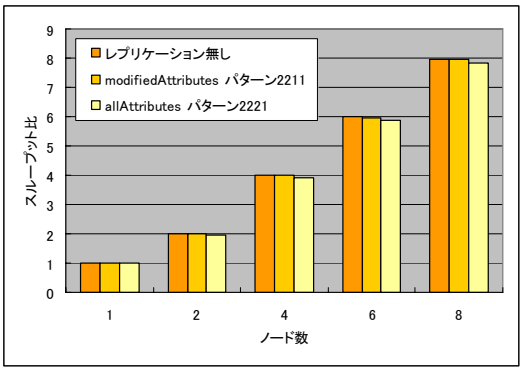


グラフ 10-20
転送タイミング(トリガー)によるレスポンスタイムの違い(中負荷 700 スレッド)

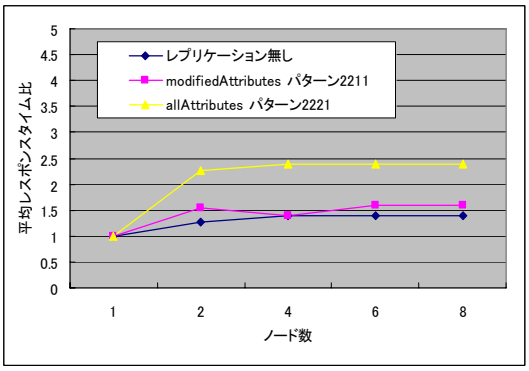
10-3-3. 転送グループ(スコープ)による性能への影響

中負荷

グラフ 10-21 とグラフ 10-22 は、中負荷において転送グループ(スコープ)を modifiedAttributes、allAttributes と設定した際のスループットとレスポンスタイムの比較である。中負荷な条件では、それぞれの設定においてスループットには違いがない。レスポンスタイムは allAttributes の方が数%程度劣化している。

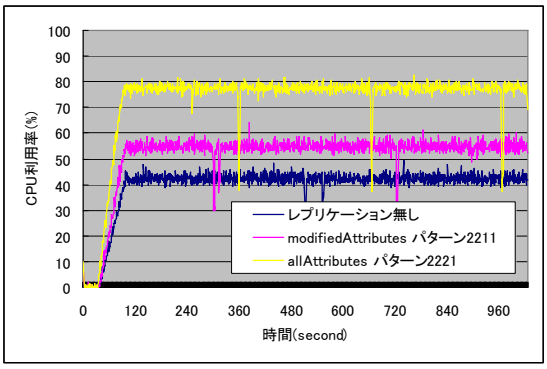


グラフ 10-21
転送グループ(スコープ)によるスループットへの影響(中負荷 350 スレッド)

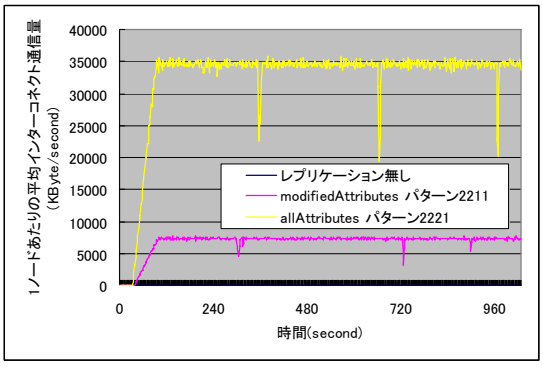


グラフ 10-22
転送グループ(スコープ)によるレスポンスタイムへの影響(中負荷 350 スレッド)

次に、CPU 使用率の違いをグラフ 10-23 で、インターコネクト転送量の違いをグラフ 10-24 で示す。allAttributes は modifiedAttributes と比べて CPU 使用率が 25%程度増加しており、非常にオーバーヘッドが大きいことがわかる。インターコネクト転送量も 27Mbytes/s 程度増加している。これは、allAttributes ではレプリケーションが行われるたびにそのセッションがそれまでに設定した全てのセッション・オブジェクトを転送するためである。従って、アプリケーションで生成されるセッション・オブジェクトの数およびサイズに依存してオーバーヘッドが変わる。



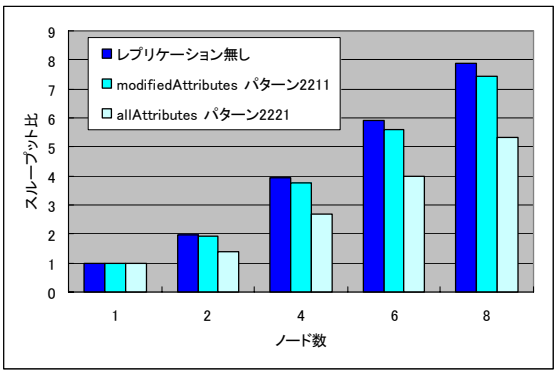
グラフ 10-23
転送グループ(スコープ)による CPU 使用率の違い(中負荷 350 スレッド)



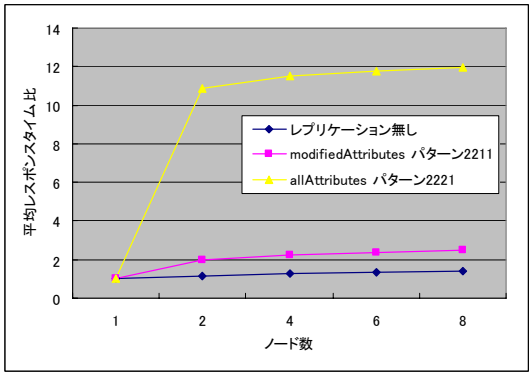
グラフ 10-24
転送グループ(スコープ)によるインターコネクト転送量の違い(中負荷 350 スレッド)

高負荷

グラフ 10-25 とグラフ 10-26 は、高負荷において転送グループ(スコープ)を modifiedAttributes、allAttributes と設定した際のスループットとレスポンスタイムの比較である。allAttributes のオーバーヘッドが顕著に現れており、約 30%スループットが低下している。また、レスポンスタイムは 8 倍程度伸びている。中負荷においては、CPU 使用率に影響していたオーバーヘッドが、高負荷な場合にはスループットやレスポンスタイムに大きく影響していることがわかる。

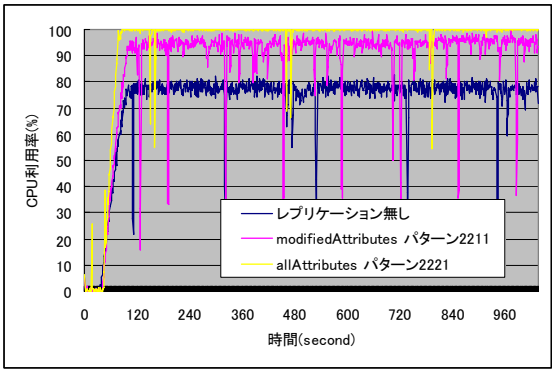


グラフ 10-25
転送グループ(スコープ)によるスループットへの影響(高負荷 700 スレッド)

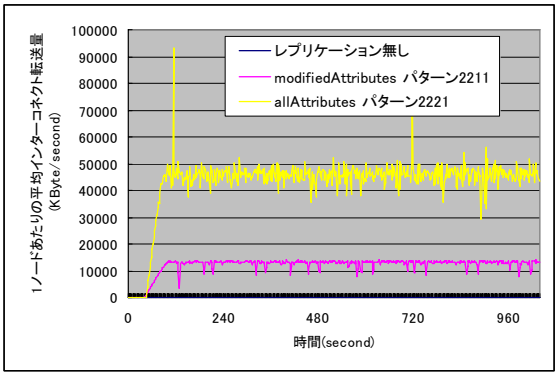


グラフ 10-26
転送グループ(スコープ)によるレスポンスタイムへの影響(高負荷 700 スレッド)

次に、グラフ 10-27 とグラフ 10-28 で CPU 使用率の違いおよびインターコネクト転送量の違いを示す。allAttributes は CPU 使用率がほぼ 100%で張り付いており、完全に CPU ボトルネックとなっている。インターコネクト転送量も 33Mbytes/s 程度増加しており、中負荷と同様に非常にオーバーヘッドが大きいことがわかる。



グラフ 10-27
転送グループ(スコープ)によるスループットへの影響(高負荷 700 スレッド)



グラフ 10-28
転送グループ(スコープ)によるインターコネクト転送量の違い(高負荷 700 スレッド)

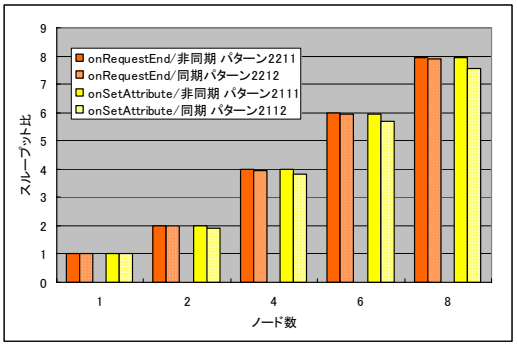
10-3-4. 同期転送による性能への影響

中負荷

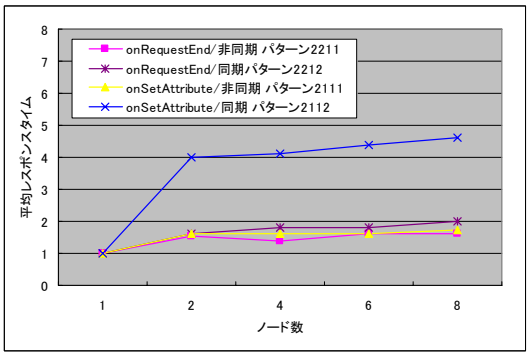
グラフ 10-29 とグラフ 10-30 は、中負荷において、onRequestEnd および onSetAttribute の両ケースでレプリケーション転送の同期、非同期による性能への影響を示したものである。また、グラフ 10-31 とグラフ 10-32 はそれぞれ、その際のインターコネクト転送量と CPU 使用率を比較したものである。

onRequestEnd では、中負荷において、同期と非同期においてスループットおよびレスポンスタイムへの影響の違いはほとんどなかった。同期モードでは非同期に比べインターコネクト転送量が 500～700Kbyte/秒程度多いことが分かる。このため、同期モードの CPU 使用率は 5%ほど高い。

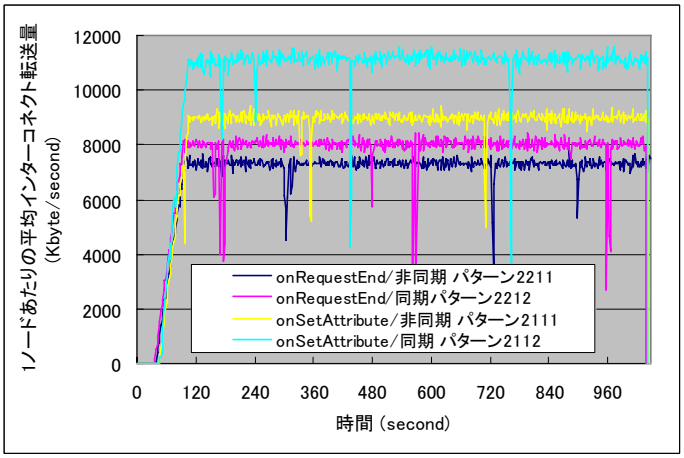
onSetAttribute では、中負荷においても同期によるスプープットとレスポンスタイムへの影響が見られた。スループットは約 5%ダウン、レスポンスタイムは 2～2.5 倍に劣化している。同期モードでは、HTTP リクエストの処理中に setAttribute()がコールされるタイミングでレプリケーションの受信確認が行われるため、その待機時間がレスポンスタイムに影響していると考えられる。同期モードでは、インターコネクト転送量は 2Mbyte/秒程度多く、CPU 使用率も 15%ほど高い。



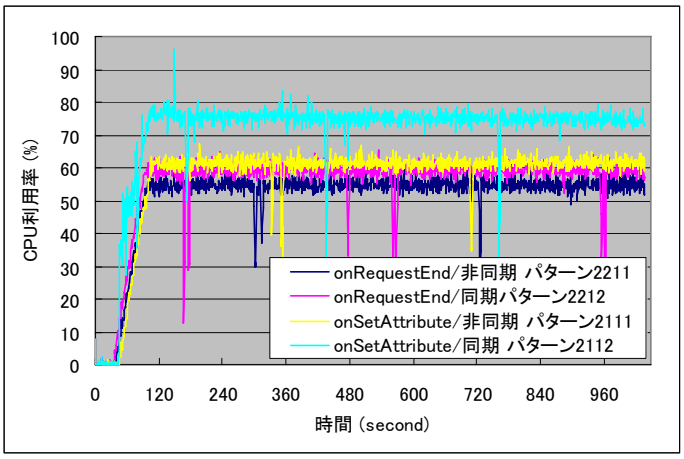
グラフ 10-29
同期転送によるスループットへの影響(中負荷 350 スレッド)



グラフ 10-30
同期転送によるレスポンスタイムへの影響(中負荷 350 スレッド)



グラフ 10-31 同期転送によるインターコネクト転送量の違い(中負荷 350 スレッド)

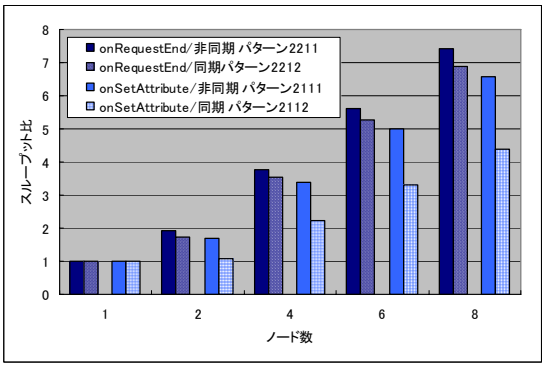


グラフ 10-32 同期転送による CPU 使用率の違い(中負荷 350 スレッド)

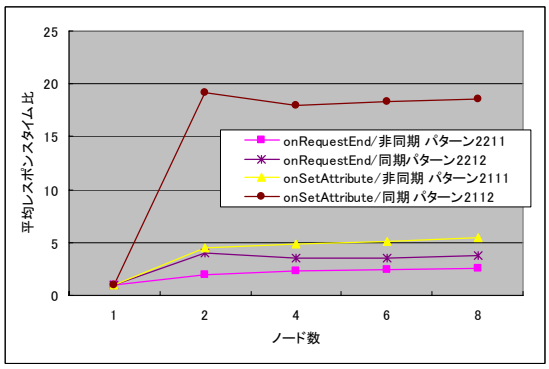
高負荷

グラフ 10-33 とグラフ 10-34 は、高負荷において onRequestEnd および onSetAttribute の両ケースでレプリケーション転送の同期、非同期による性能への影響を示したものである。中負荷と同様、「onRequestEnd & 非同期」、「onRequestEnd & 同期」、「onSetAttribut&非同期」、「onSetAttribute&同期」の順番でスループットとレスポンスタイムへの影響が大きくなっていることがわかる。しかし、「onSetAttribute&同期」のケースは他のケースと比べてスループットへの影響がより大きかった。また、スループットの実測値も中負荷から高負荷にかけて、ほとんど増加しなかった。

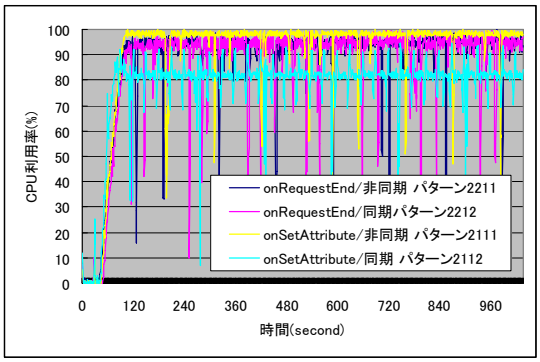
グラフ 10-35 とグラフ 10-36 の CPU 利用率とインターコネクト転送量を見ると、高負荷における「onSetAttribute&同期」のケース以外は CPU 使用率とインターコネクト転送量が増加している。「onSetAttribute&同期」では、同時実行スレッド数が 350 の場合でも CPU 使用率が約 80%と、他のケースに比べて負荷が高い状態であったが、同時実行スレッド数が 700 の場合でも、CPU 使用率やネットワーク転送量がほとんど変わっておらず、結果として同時実行スレッド数を増やしてもスループットがほとんど向上しなかった。CPU リソースを使い切ることなくスループットが頭打ちになっていることから、「onSetAttribute&同期」では、同期処理における待機のボトルネックが顕著に現れていると考えられる。



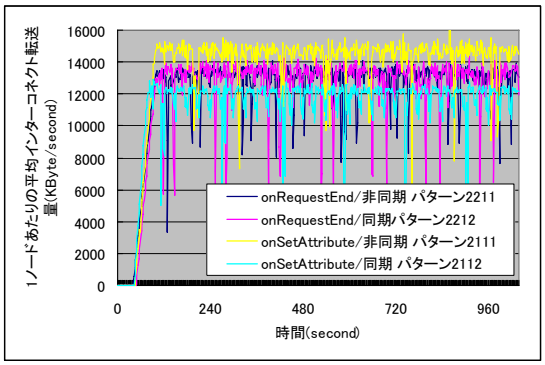
グラフ 10-33
同期転送によるスループットへの影響(高負荷 700 スレッド))



グラフ 10-34
同期転送によるレスポンスタイムへの影響(高負荷 700 スレッド)



グラフ 10-35
同期転送による CPU 使用率の違い(高負荷 700 スレッド)



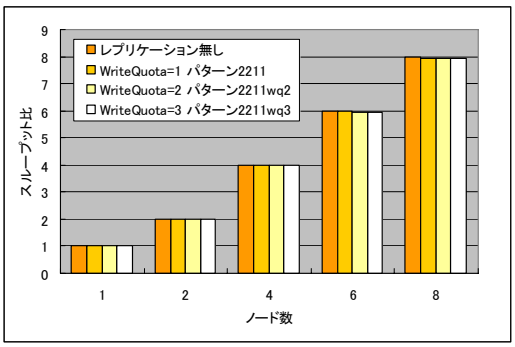
グラフ 10-36
同期転送によるインターコネクト転送量の違い(高負荷 700 スレッド)

10-3-5. write-quota 増加による性能への影響

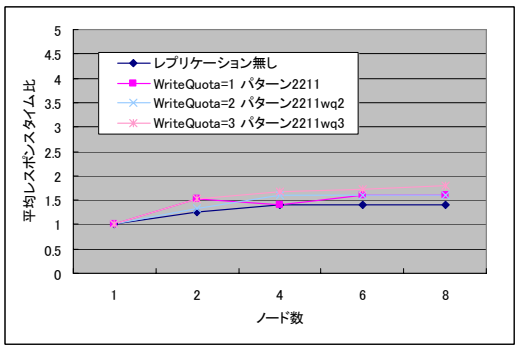
中負荷

グラフ 10-37 とグラフ 10-38 はそれぞれ、中負荷において write-quota の数を増加させたときのスループットとレスポンスタイムを測定した結果である。中負荷においては、write-quota の増加によってスループットとレスポンスタイムに影響はほとんどない。特に 2 ノード構成においては、write-quota の設定が 1、2、3 のいずれでも、自分以外の 1 つのノードにのみセッションオブジェクトをレプリケートするので、性能結果に全く違いがなかった。

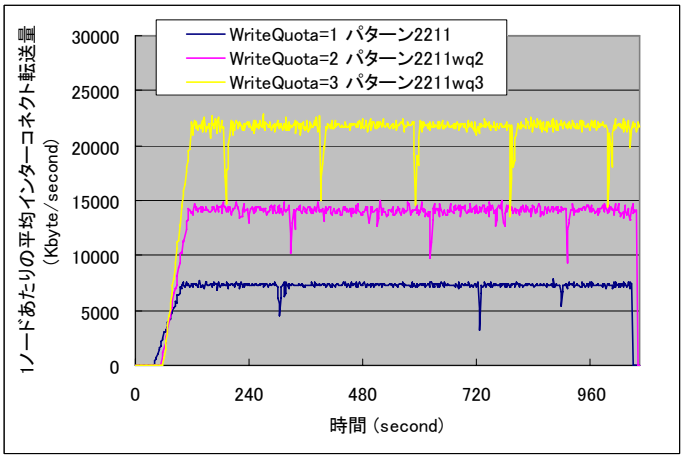
グラフ 10-39 とグラフ 10-40 は、write-quota の違いにおけるインターコネクト転送量と CPU 使用率を比較したグラフである。Write-quota の値に比例してインターコネクト転送量が 2 倍、3 倍となっている。CPU 使用率は write-quota が増えるに従って約 5%づつ増加していることが分かる。



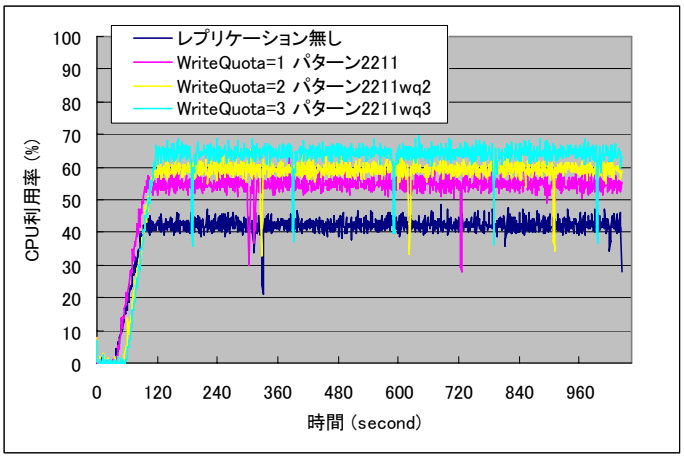
グラフ 10-37
write-quota によるスループットへの影響(中負荷 350 スレッド)



グラフ 10-38
write-quota によるレスポンスタイムへの影響(中負荷 350 スレッド)



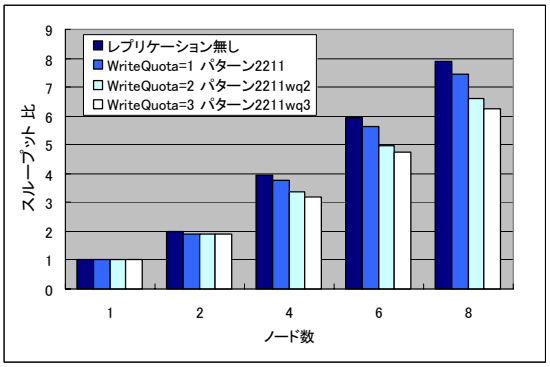
グラフ 10-39　write-quota によるインターコネクト転送量の違い(中負荷 350 スレッド)



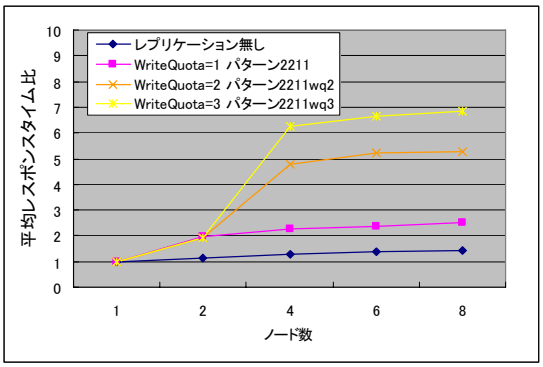
グラフ 10-40 write-quota による CPU 使用率の違い(中負荷 350 スレッド)

高負荷

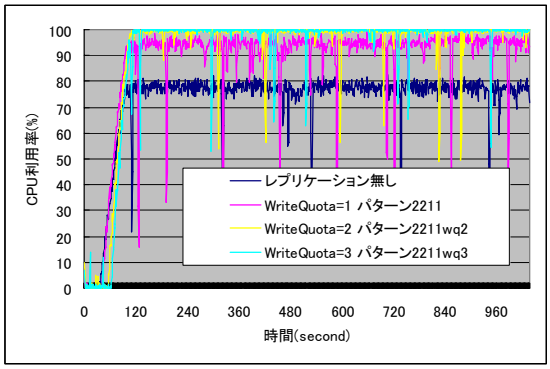
グラフ 10-41 とグラフ 10-42 は、それぞれ高負荷において、write-quota の数を増加させたときのスループットとレスポンスタイムを測定した結果である。2 ノード構成では結果に違いがないが、4 ノードから 8 ノード構成においては、write-quota 数の増加に伴い性能への影響が見られる。Write-quota が 1 から 2 に増加すると約 10%、2 から 3 に増加すると約 5%のスループットへの影響が見られた。また、write-quota が 2 および 3 のケースでは、1 ノード上の OC4J インスタンスが一つでは、JVM のヒープ領域が不足し OutOfMemory エラーが発生することがあった。これは、本環境では一つの JVM が最大 2GB までしかヒープサイズを確保できないが、write-quota の設定が増えるにつれ保持しなければならない他ノードのセッション・オブジェクトが多くなり、ヒープ領域を圧迫してためである。1 ノード上で起動する OC4J インスタンスを増やすことで、総ヒープ領域を増やすことができ、エラーの発生を防ぐことができた。4 ノード、6 ノード、8 ノード構成においては、各ノードにおいて OC4J インスタンスを 2 つ起動している。



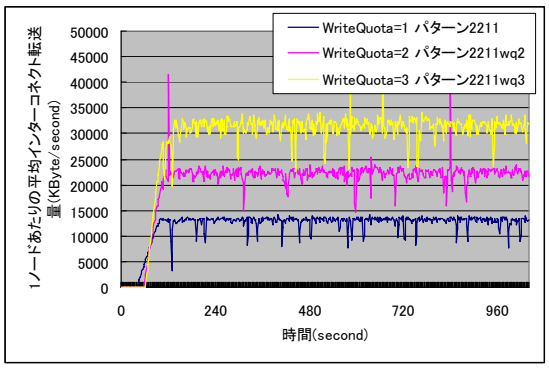
グラフ 10-41
write-quota によるスループットの違い(高負荷 700 スレッド)



グラフ 10-42
write-quota によるレスポンスタイムの違い(高負荷 700 スレッド)



グラフ 10-43
write-quotaによるCPU使用率の違い(高負荷 700 スレッド)



グラフ 10-44
write-quota にインターコネクト転送量の違い(高負荷 700 スレッド)

11. ベストプラクティス

本検証の結果から、性能面でのベストプラクティスを示す。なお、ここで説明するのは最も高い性能を出すことができる設定であるが、場合によっては可用性要件を実現できないかもしれない。最終的には、可用性要件とこれまで報告した性能検証を照らし合わせ、最適なものを選択してほしい。

JVM の起動オプション　-XX:+UseAgressiveHeap

JVM の起動オプションとして、Aggressive Heap オプションを指定することで、JVM のガベージコレクションによる性能への影響を最小限にすることができる。

レプリケーション転送方式 Peer-to-Peer

Peer-to-Peer と Multicast 転送方式で、アプリケーションの性能および可用性に与える影響にほとんど違いがない。今回検証の中で、高負荷な条件では、Multicast 方式では UDP プロトコルを使用していることに起因すると思われる、性能上不安定な動作が見られることがあった。本ホワイトペーパーでは、高負荷でも安定した性能を出した Peer-to-Peer の選択をベストプラクティスとする。

転送タイミング onRequestEnd

onRequestEnd を指定することで、オーバーヘッドを低く抑えることができる。

転送グループ modifiedAttributes

modifiedAttributes を指定することで、オーバーヘッドを低く抑えることができる。

同期または非同期

非同期に指定することで、オーバーヘッドを低く抑えることができる。

ネットワーク構成

クラスタ構成におけるノード数が増えるに従い、ネットワーク・トラフィックも増大する。そのため、セッション・レプリケーション用に専用のプライベート・ネットワークを用意することが望ましい。またクライアントとの接続ネットワークは、VLAN あるいは Link Aggregation で帯域を確保することが望ましい。

12. まとめ

今回の検証によって、日立の BladeSymphony プラットフォームと Oracle のグリッドインフラストラクチャの組み合わせにおいて非常に高いシステム性能拡張性を実証することができたことは大きな成果である。また、アプリケーション・サーバー層とデータベース・サーバー層を含めた WEB 3 層構成上の Java Web アプリケーションという、エンタープライズ領域で一般的に採用されつつあるアーキテクチャでの大規模性能実績としても重要な意味を持つ。システムの性能拡張にあたってはさまざまな点を考慮する必要があり、特に大規模構成においはネットワーク構成やストレージなどのハードウェア設計の観点も非常の重要な点になってくるが、日立 BladeSymphony であれば、Oracle Application Server 10g と Oracle Real Application Clusters 10g それぞれ最大 8 ノードクラスタ構成という大規模なシステムにおいても十分適用可能であることを理解していただけたと思う。

また、このような大規模構成において、Oracle Application Server 10g 特有のノード間メモリ通信による HTTP セッション・レプリケーション機能を徹底的に検証することで、本機能の有効性と各レプリケーション方式による特徴を把握することができた。性能と可用性の両方を最大限に活用するためのベストプラクティスをまとめてあるので、システム設計時の指針として活用していただければ幸いである。

注１） JPetStore および本書で掲載している画面イメージは、Spring Framework 1.2.8(http://www.springframework.org/)に付属するサンプルアプリケーションを使用しています。