IdPClusteringPerformance

Shibboleth-IdP冗長化パフォーマンス比較試験報告書

2012年1月17日
国立情報学研究所

※ Stateless Clustering方式は、SAML2を想定しているためCryptoTransientIDは不使用。使用するとパフォーマンスが悪くなる可能性あり。
※ Terracottaによる冗長化について、EventingMapBasedStorageServiceにConcurrentDistributedMapを使うなどの最適化は行っていない。

- 目次 -

1. はじめに

1.1. 目的

本書は、冗長化したShibboleth-IdPのパフォーマンス計測の結果報告書です。
本書にてパフォーマンス計測のシステム構成と計測内容、計測結果を記載します。

1.2. 比較する Shibboleth-IdP冗長化方式

パフォーマンス計測の対象となる冗長化方式は下記とします。

- Terracotta方式
  - 認証情報は、冗長化したサーバー間で同期されるため、tomcatおよびTerracottaの方系がダウンしても再認証は求められない。
  - 参考ページ
    https://wiki.shibboleth.net/confluence/display/SHIB2/IdPCluster
- memcached方式（No Replication）
  - 認証情報は、ログインしたtomcatと設定された各memcachedサーバーに認証情報を分割し振り分けて保管する。memcachedの方系がダウンした場合は、ログインしたことのあるサーバーへのアクセスではtomcatに認証情報が残っていれば、再認証は求められない。しかし、ログインしていないサーバーへのアクセス、もしくは、tomcatに認証情報が残っていない状態でのアクセスでは、再認証が求められる。
  - 参考ページ
    https://wiki.shibboleth.net/confluence/display/SHIB2/Memcached+StorageService
- repcached方式（replication）
  - 認証情報は、冗長化したサーバー間で同期されるため、tomcatおよびrepcachedの方系がダウンしても再認証は求められない。

    - 全サーバーが停止した場合、認証情報が失われるので、再認証が求められる。
    - 参考ページ
      https://wiki.shibboleth.net/confluence/display/SHIB2/Memcached+StorageService
- IdP Stateless Clustering方式
    - 認証情報暗号化しCookieに入れてブラウザが持つ。サーバーに依存しないため、サーバーがダウンしても再認証は求められない。
    - 参考ページ
      https://wiki.shibboleth.net/confluence/display/SHIB2/IdPStatelessClustering

1.3. 検証環境の構成

本試験における検証環境は下記に記すサーバーにて計測する。各サーバーのスペックは以下に示します。
表 1-1 検証に使用したサーバー一覧

No	サーバー	環境
1	SP　& 負荷テストツール	OS : CentOS 5.7 64bit CPU : Intel Xeon 2.27GHz Memory : 2GB No2～No7 は 同一サーバーにて VMware で動作。
2	IdP Terracotta 1、LDAP	
3	IdP Terracotta 2	
4	IdP memcached 1、LDAP repcached　1	
5	IdP memcached 2、repcached　2	
6	IdP StatelessClustering 1 & LDAP	
7	IdP StatelessClustering 2	

表 1-2 各検証に使用したソフトウェアのバージョン

No	ソフトウェア	バージョン	対象サーバー	備考
1	Apache HTTP Server	2.2.3	1,2,3,4,5,6,7	
2	Shibboleth-SP	2.4.3-2.2	1	
3	Shibboleth-IdP	2.3.5	2,3,4,5,6,7	
4	memcached	1.4.10	4,5	メモリーの割り当ては512MB
5	repcached	2.2	4,5	メモリーの割り当ては512MB repcached に使われている memcachedのバージョンは1.2.8
6	Osuidpext	1.1	6,7	Stateless Clustering用ライブラリ
7	Terracotta	3.6.0	2,3	
8	JDK	1.6.0_30	2,3,4,5,6,7	
9	Apache Tomcat	7.0.23	2,3,4,5,6,7	
10	OpenLDAP	2.3.43	2,4,6	各IdPサーバーの1台に構築

※ 対象サーバーに記述する値は、表 1 1 検証に使用したサーバー一覧 に記載するNoです。

2. パフォーマンスの測定方法

2.1. クライアント環境

- 検証環境と同一ネットワーク内にあるSPサーバーに負荷テストツールを置き、各計測対象サーバーへ負荷をかけてパフォーマンスを測定します。

2.2. ロードバランシング

- 冗長化した各サーバーへのロードバランシングは、計測対象ではない冗長化方式のサーバーにApacheモジュールのmod_proxy_balancer を使いリクエスト毎に分散する方式とします。

2.3. 内容

- 負荷テストツールを使い、下記のパターンでの測定を行います。
  - パターン１：　2スレッド×1000回
  - パターン２：　10スレッド×200回
  - パターン３：　20スレッド×200回
- 助走期間として、各測定前に１分程度負荷ツールを実行します。
- Teracottaのみ各測定後に再起動を実施します。
- 計測は、１認証処理にかかる時間、および１秒間の認証処理回数を計測します。
- スレッドは毎アクセス新しいセッションを張る方法で行います。（常に認証処理が走る）
- 認証に利用するアカウントはひとつとします。

3. ツールと使用データ

3.1. 測定ツール

- Grinder
  Grinder（グライダー）は、オープンソースの負荷テストツールです。Jythonで認証の動作を書いて実行します。
  参考：https://wiki.shibboleth.net/confluence/display/SHIB2/IdPProdLoadTest

3.2. テストデータ

- 認証アカウントは学認で公開されているサンプルユーザをLDAPに設定しています。
  GakuNinShibInstall:OpenLDAPの設定

4. テスト

4.1. テスト内容

各方式で冗長化構成が機能することを確認するテスト内容を下記に示します。

テスト１	それぞれのサーバーについて、テストアカウントで認証できること、および認証結果がSPへ送信されること
テスト２	一方のサーバーで認証後、他方のサーバーに認証要求を送った場合、認証済みとして正常に処理されること
テスト３	未認証の状態で一方のサーバーでパスワード要求画面（いわゆるログイン画面）を表示した後、他方のサーバーにその応答を送った場合、正常に処理されること ※1 ※1 ブラウザによっては、ログイン画面を表示したサーバーのDNSがキャッシュされるため、他方のサーバーへ送信しようとしても、ログイン画面を表示したサーバーに応答を送ってしまいます。 Firefox9、IE9では、DNSをキャッシュし、Chrome16ではDNSをキャッシュしませんでした。 今回は、ログイン画面を表示後にブラウザのDNSキャッシュをクリアしてから、他方のサーバーに応答を送る手順で行っています。 また、今回のテストでは、DNSサーバーを用いずに、hostsファイルでサーバーの切り替えを行っています。

4.2. テスト結果

各方式でのテスト結果を下記に示します。
表 4-1 テスト結果

方式	テスト１	テスト２	テスト３

Terracotta	○	○	○
memcached	○	○	×
Stateless Clustering	○	○	×

5. パフォーマンス測定結果

5.1. Terracotta方式

5.1.1. パターン１（2スレッド×1000回）

表4-1 Terracotta方式 パターン１（2スレッド×1000回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
T1-1	2000	0	154.45	12.77
T1-2	2000	0	163.51	12.02
T1-3	2000	0	146.48	13.49
平均	2000	0	154.81	12.76

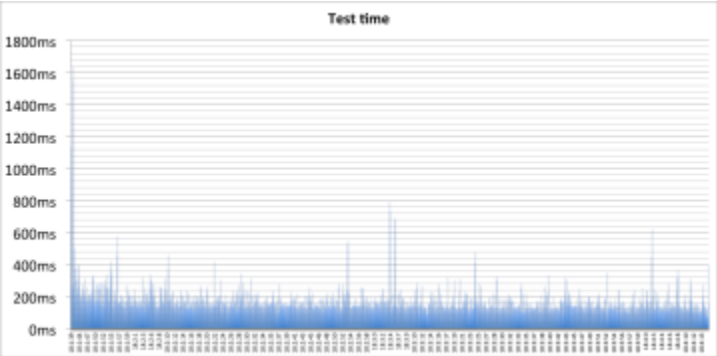

図 4-1 #T1-1の測定結果グラフ

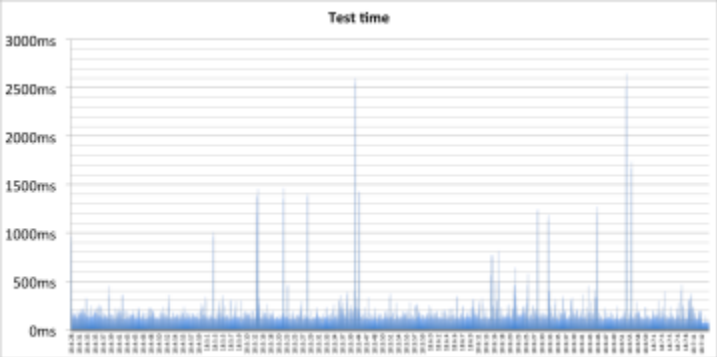

図 4-2 #T1-2の測定結果グラフ

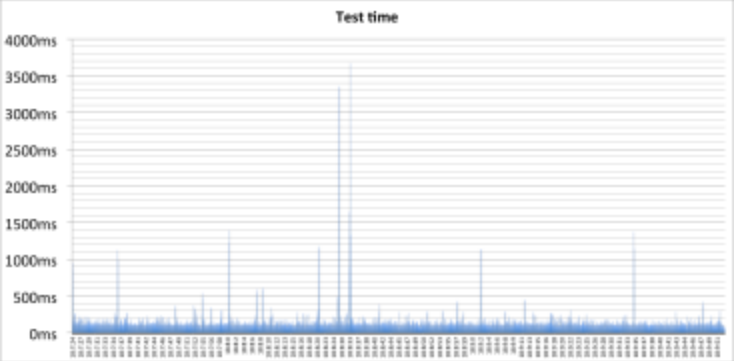

図 4-3 #T1-3の測定結果グラフ

5.1.2. パターン2（10スレッド×200回）

表4-2 Terracotta方式 パターン２（10スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
T2-1	2000	0	649.73	14.91
T2-2	2000	0	600.70	15.50
T2-3	2000	0	620.43	15.64
平均	2000	0	623.62	15.35

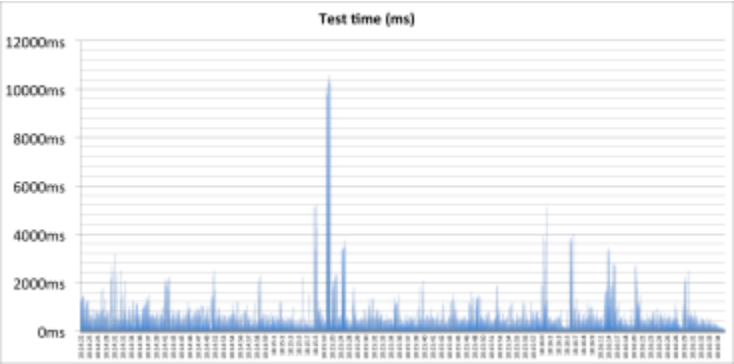

図 4-4 #T2-1の測定結果グラフ

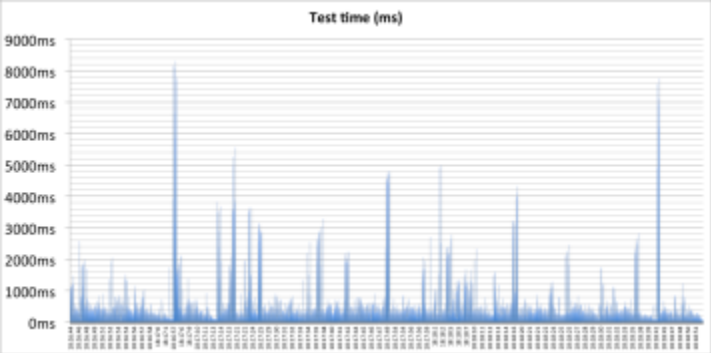

図 4-5 #T2-2の測定結果グラフ

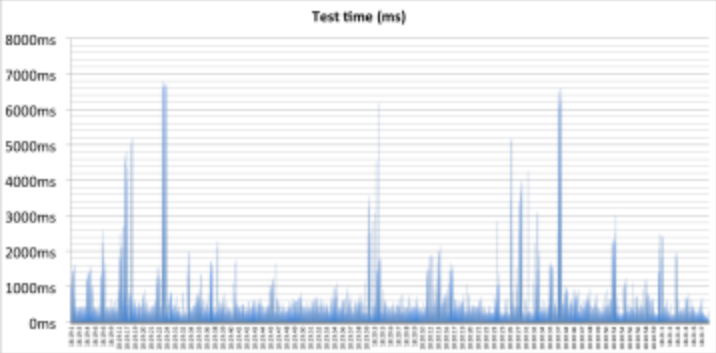

図 4-6 #T2-3の測定結果グラフ

5.1.3. パターン3（20スレッド×200回）

表4-3 Terracotta方式 パターン3（20スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
T3-1	4000	0	1212.99	15.14
T3-2	4000	0	1280.92	14.98
T3-3	4000	0	1214.07	15.55
平均	4000	0	1235.99	15.22

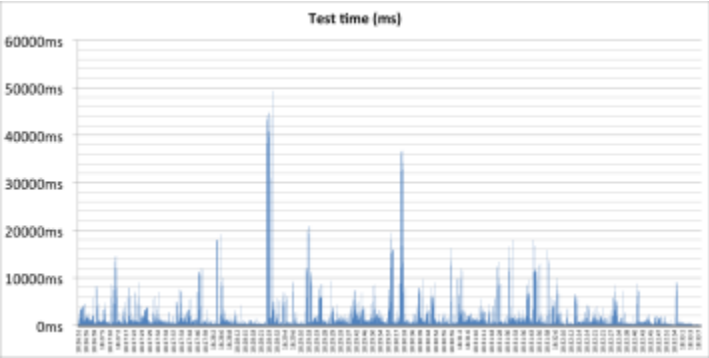

図 4-7 #T3-1の測定結果グラフ

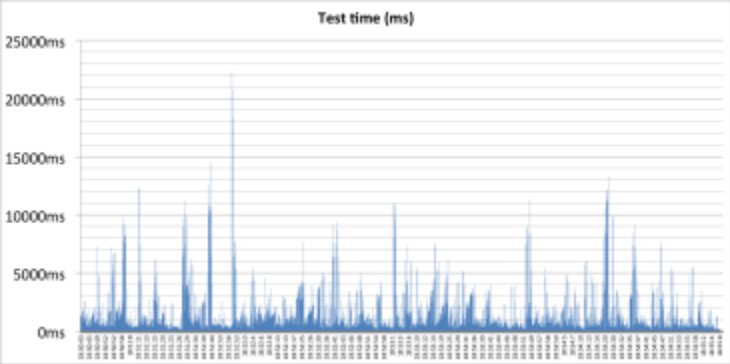

図 4-8 #T3-2の測定結果グラフ

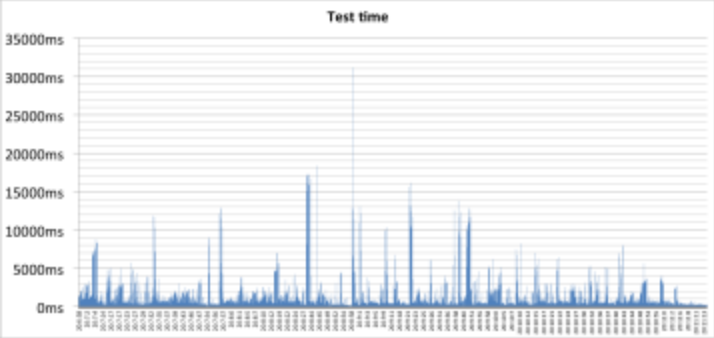

図 4-9 #T3-3の測定結果グラフ

5.2. memcached方式

5.2.1. パターン1（2スレッド×1000回）

表4-4　memcached方式　パターン１（2スレッド×1000回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
M1-1	2000	0	111.97	17.54
M1-2	2000	0	104.96	18.70
M1-3	2000	0	100.73	19.38
平均	2000	0	105.88	18.54

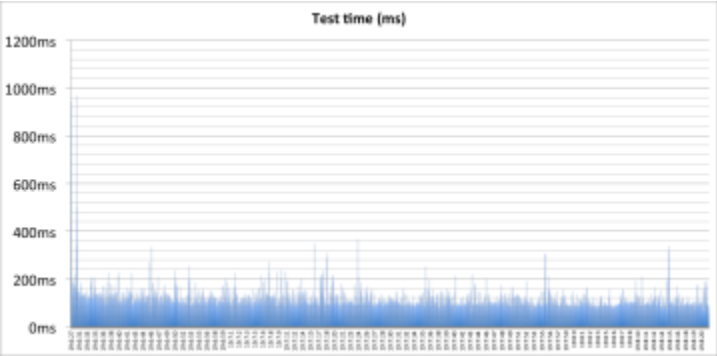

図 4-10 #M1-1の測定結果グラフ

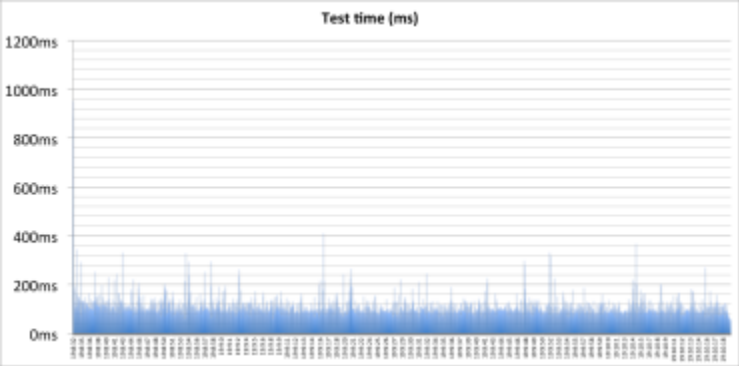

図 4-11 #M1-2の測定結果グラフ

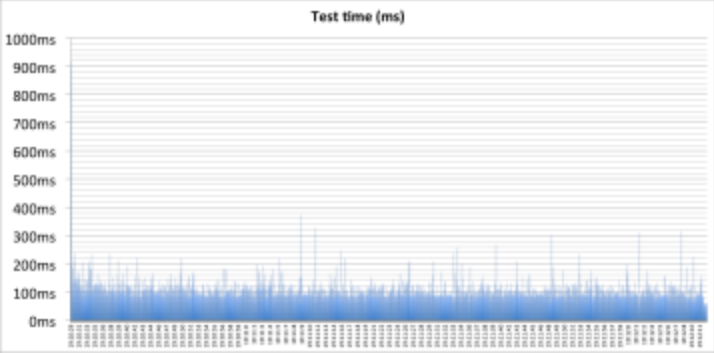

図 4-12 #M1-3の測定結果グラフ

5.2.2. パターン2（10スレッド×200回）

表4-5 memcached方式 パターン２（10スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
M2-1	2000	0	308.50	31.11
M2-2	2000	0	304.90	31.34
M2-3	2000	0	296.85	32.11
平均	2000	0	303.41	31.52

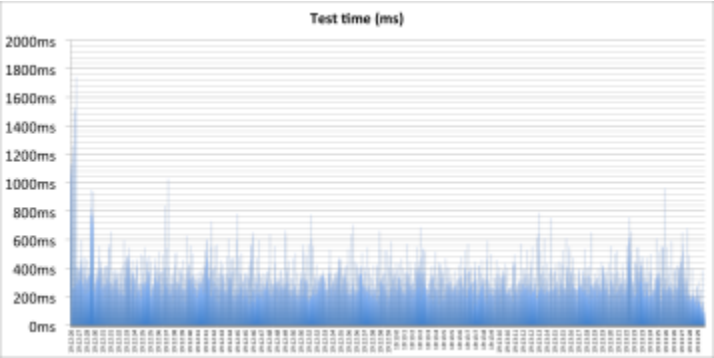

図 4-13 #M2-1の測定結果グラフ

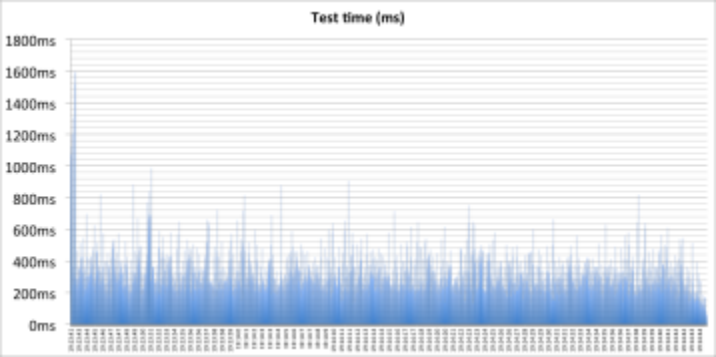

図 4-14 #M2-2の測定結果グラフ

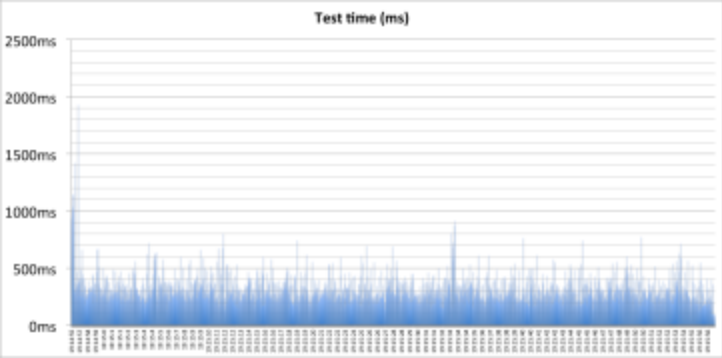

図 4-15 #M2-3の測定結果グラフ

5.2.3. パターン3（20スレッド×200回）

表4-6 memcached方式 パターン3（20スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
M3-1	4000	0	493.24	39.01
M3-2	4000	0	487.27	39.59
M3-3	4000	0	484.07	39.79
平均	4000	0	488.19	39.46

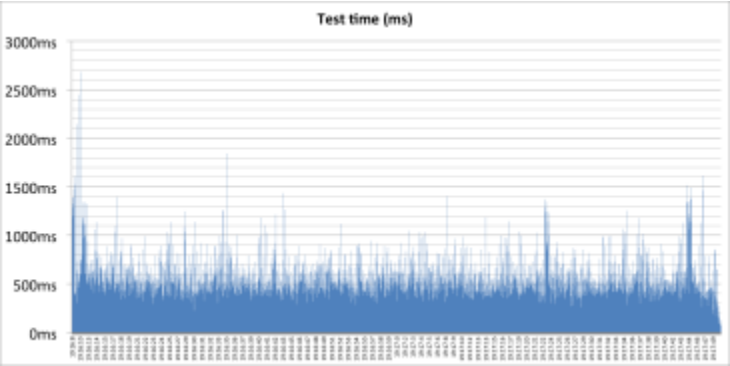

図 4-16 #M3-1の測定結果グラフ

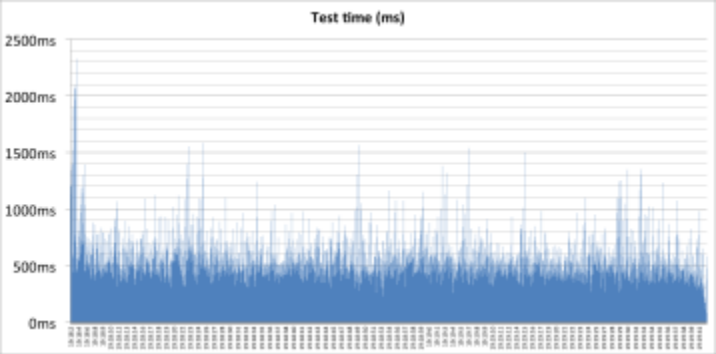

図 4-17 #M3-2の測定結果グラフ

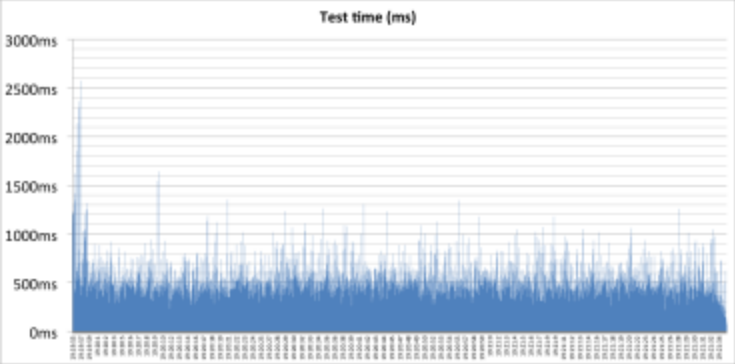

図 4-18 #M3-3の測定結果グラフ

5.3. repcached方式

5.3.1. パターン1（2スレッド×1000回）

表4-7 repcached方式 パターン１（2スレッド×1000回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
R1-1	2000	0	106.56	18.40
R1-2	2000	0	98.74	19.74
R1-3	2000	0	97.84	19.93
平均	2000	0	101.04	19.35

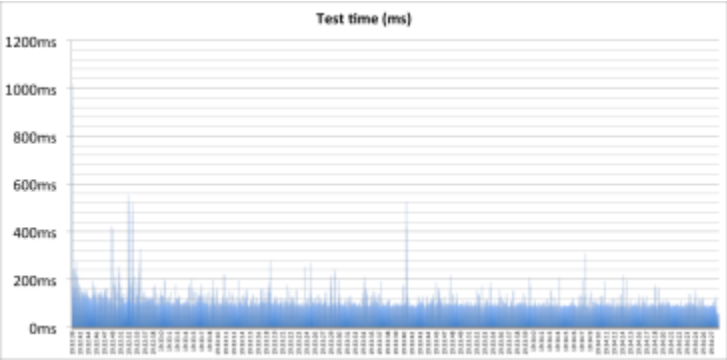

図 4-19 #R1-1の測定結果グラフ

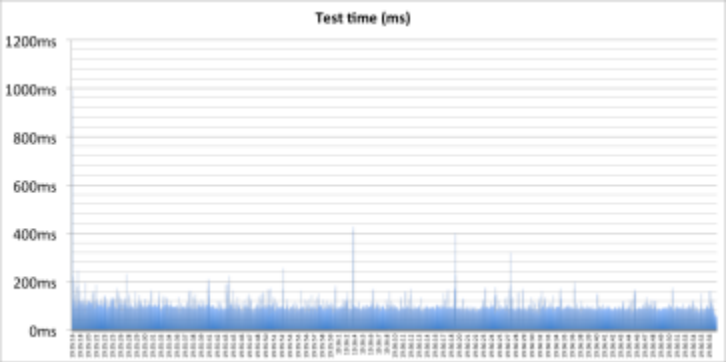

図 4-20 #R1-2の測定結果グラフ

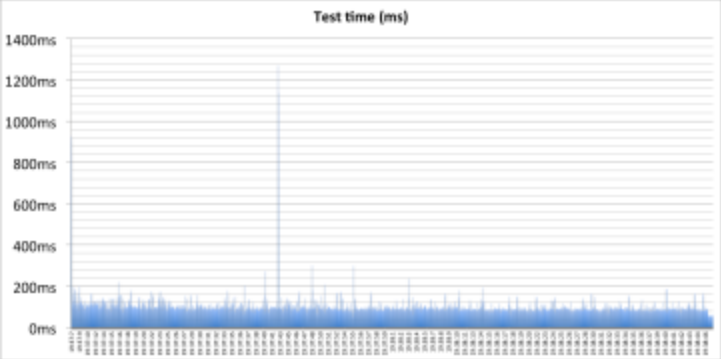

図 4-21 #R1-3の測定結果グラフ

5.3.2. パターン2（10スレッド×200回）

表4-8 repcached方式　パターン２（10スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
R2-1	2000	0	304.33	31.22
R2-2	2000	0	299.58	31.57
R2-3	2000	0	296.84	32.18
平均	2000	0	300.25	31.65

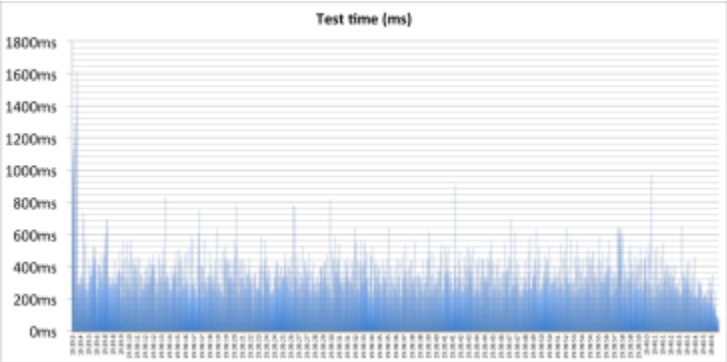

図 4-22 #R2-1の測定結果グラフ

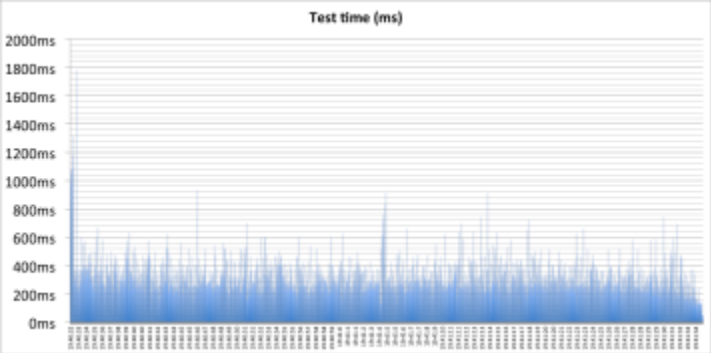

図 4-23 #R2-2の測定結果グラフ

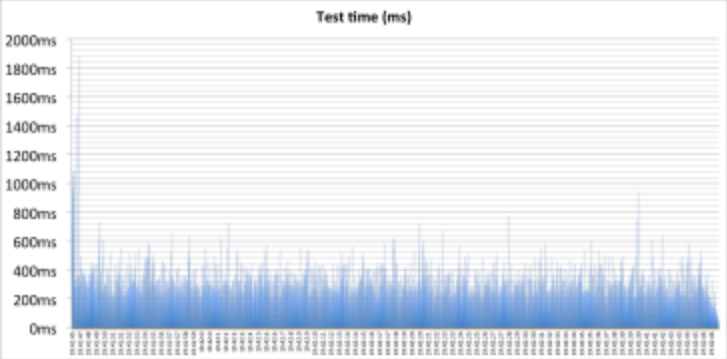

図 4-24 #R2-3の測定結果グラフ

5.3.3. パターン3（20スレッド×200回）

表4-9 repcached方式 パターン3（20スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
R3-1	4000	0	497.79	38.50
R3-2	4000	0	490.10	39.32
R3-3	4000	0	495.44	38.87
平均	4000	0	494.44	38.89

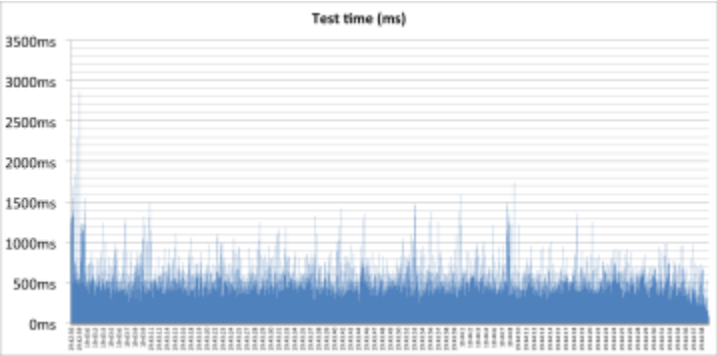

図 4-25 #R3-1の測定結果グラフ

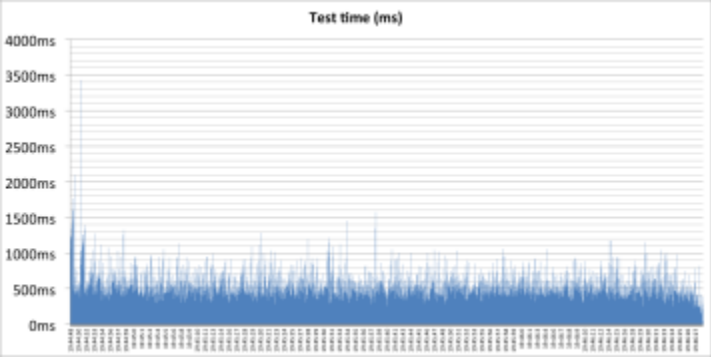

図 4-26 #R3-2の測定結果グラフ

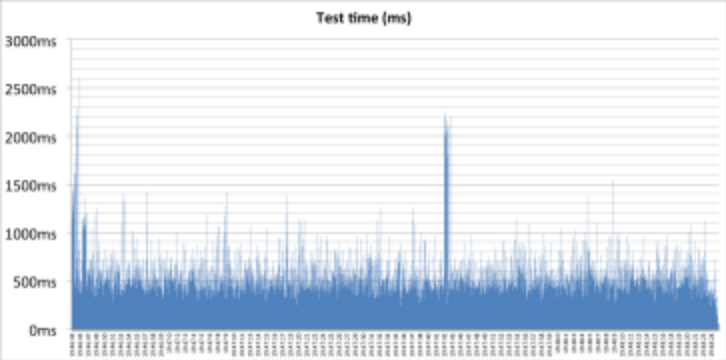

図 4-27 #R3-3の測定結果グラフ

5.4. IdP Stateless Clustering方式

5.4.1. パターン1（2スレッド×1000回）

表4-10 IdP Stateless Clustering方式 パターン１（2スレッド×1000回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
S1-1	2000	0	80.46	24.05
S1-2	2000	0	74.14	26.08
S1-3	2000	0	70.30	27.36
平均	2000	0	74.96	25.83

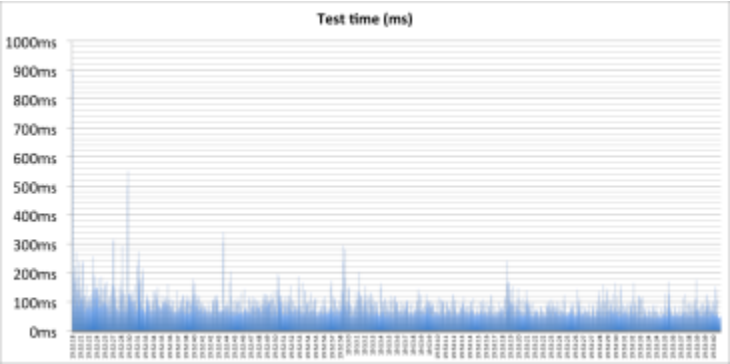

図 4-28 #S1-1の測定結果グラフ

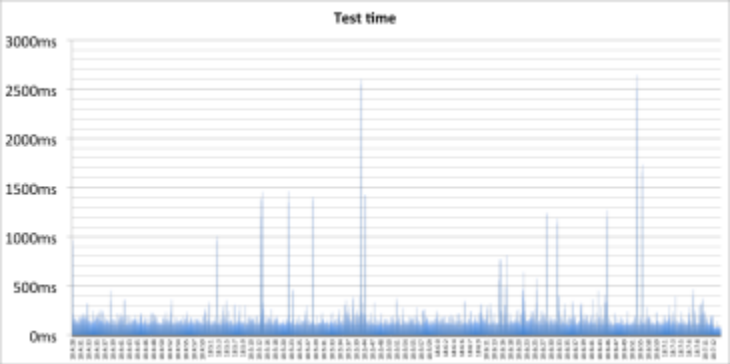

図 4-29 #S1-2の測定結果グラフ

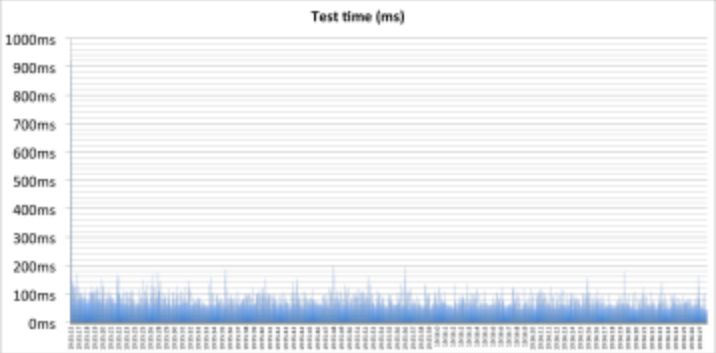

図 4-30 #S1-3の測定結果グラフ

5.4.2. パターン2（10スレッド×200回）

表4-11 IdP Stateless Clustering方式　パターン２（10スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
S2-1	2000	0	228.11	41.24
S2-2	2000	0	217.72	42.59
S2-3	2000	0	215.96	42.44
平均	2000	0	220.59	42.09

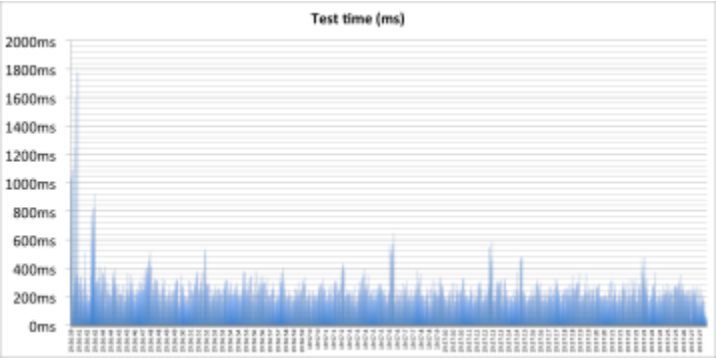

図 4-31 #S2-1の測定結果グラフ

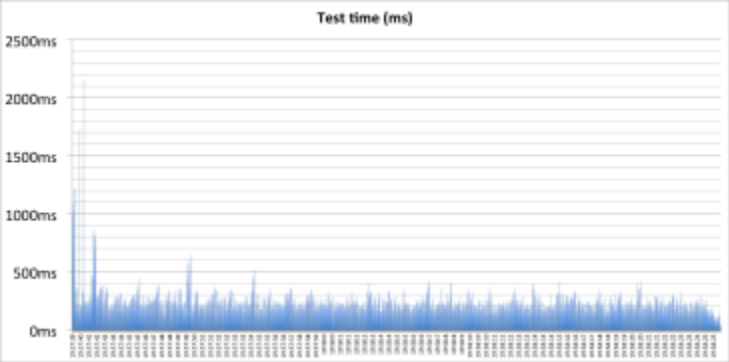

図 4-32 #S2-2の測定結果グラフ

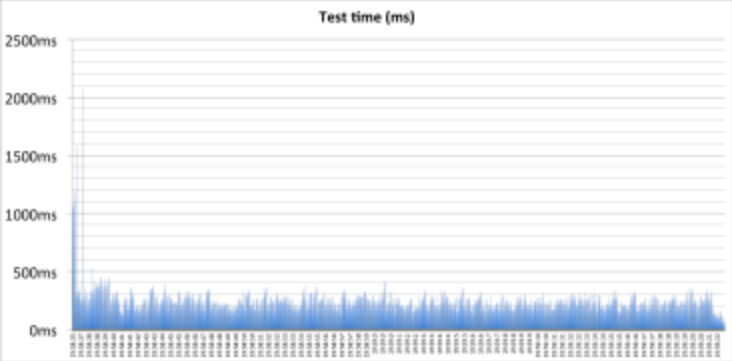

図 4-33 #S2-3の測定結果グラフ

5.4.3. パターン3（20スレッド×200回）

表4-12 IdP Stateless Clustering方式 パターン3（20スレッド×200回）

#	Tests	Errors	MeanTestTime(ms)	TPS(トランザクション／秒)
S3-1	4000	0	403.31	46.94
S3-2	4000	0	389.94	48.23
S3-3	4000	0	402.58	46.44
平均	4000	0	398.61	47.20

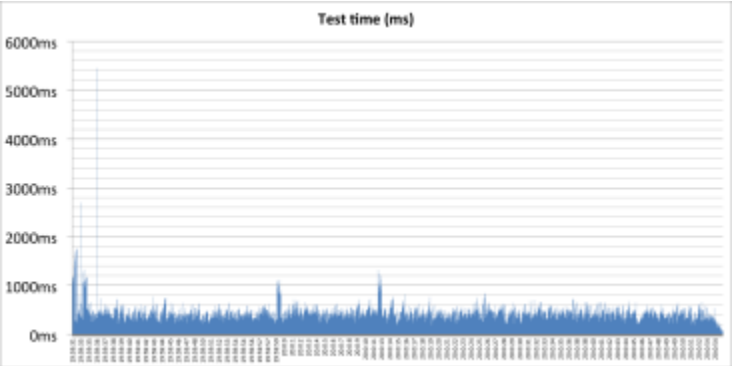

図 4-34 #S3-1の測定結果グラフ

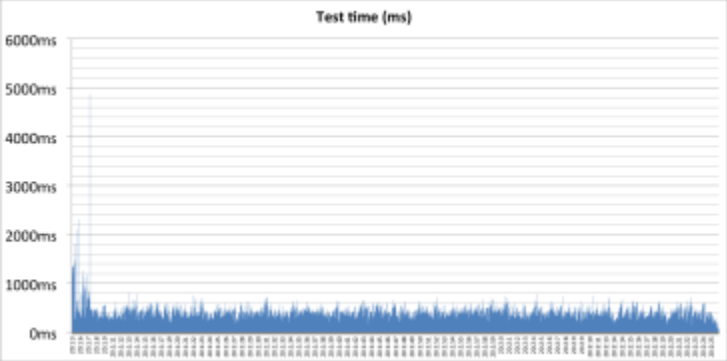

図 4-35 #S3-2の測定結果グラフ

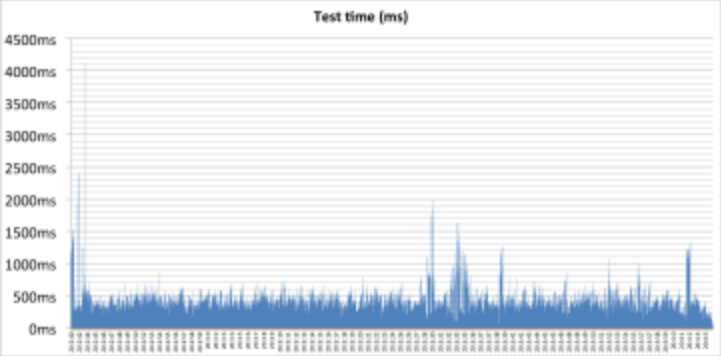

図 4-36 #S3-3の測定結果グラフ

5.5. 測定値の平均

表4-13 平均一覧

#	方式	パターン１ TPS(トランザクション／秒)平均	パターン２ TPS(トランザクション／秒)平均	パターン３ TPS(トランザクション／秒)平均
1	Terracotta	12.76	15.64	15.22
2	memcached	19.38	31.52	39.46
3	repcached	19.35	31.65	38.89
4	Stateless Clustering	25.83	42.09	47.20

6. 考察

今回行ったパフォーマンスの測定結果から判断すると同時に行う処理の数に関わらずStateless Clustering方式が他の方式より早いが、処理数が10以下と少ない場合、IdP Stateless Clustering方式、repcached方式、memcached方式とも同等の性能がでている。
なお、別紙の各環境構築手順書を見ると、Stateless Clustering方式はソースコードからjarファイルを作成する必要があり、他の方式と比べると複雑である。
Shibboleth-IdPの冗長化を導入する場合は、利用者数や構築にかけられるコストなどから判断して、利用する方式を選択することが望ましい。