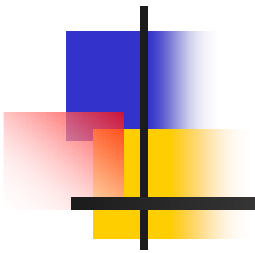
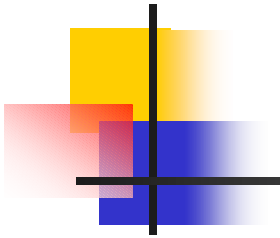


「PHP Load Test in OSDL」

桝形　誠二

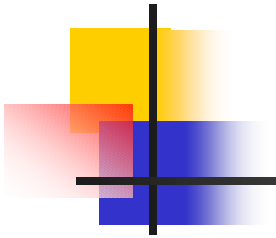


「PHP Load Test in OSDL」の目的

一般の住人方々を十分説得できるような「リアル」な実験を目的とした
大規模なアクセスや大規模な環境でのPHPの負荷テストを行い、実績を
つんでいく事を目的としています。

環境はOSDLにて、用意してもらう予定です。
実施予定は。。。未定！

なんてこった。。。個人的には年末くらいまでにはやりたいなっと。



OSDLとは？

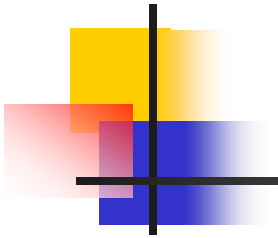
オープン ソース デベロップメント ラボ

Open Source Development Lab

大手企業によるLinux開発支援研究所で、目的は、Linuxの信頼性や可用性、実用性を向上させ、エンタープライズコンピューティングのニーズにより一層応えられるようにすること。

また、オープンソース開発者のための非営利目的なリソースラボで、機材とインフラを大規模な Linux 対応技術プロジェクトへ提供し、エンタープライズ向けアプリケーションの開発をサポートする。

Propose a Projectで説明されている概要に適合していれば、誰でも利用はできる。



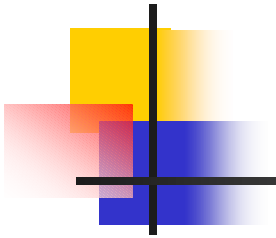
何故、このような事を行うのか？

■W大学のトラブルはSQL文のミスが原因

http://itpro.nikkeibp.co.jp/members/SI/ITARTICLE/20030530/1/

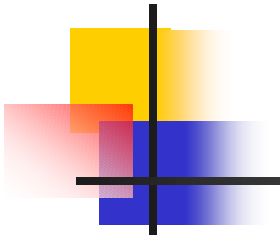
■PHP-Users MLでもかなりの反響が。

http://ns1.php.gr.jp/pipermail/php-users/2003-June/thread.html#16298



素人ながらの素朴な疑問

- いくら不慣れであったとしても、あのN★Cですら失敗したシステム構成。
- 果たして、PHPは大規模なアクセスや大規模な環境では適用できないのか？運用していくのは難しいのか？



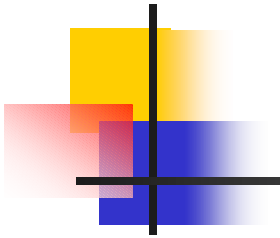
いいえ！決して、そんな事はありません。と強く言い切ってみる。

■確かに、素のPHPでは多少厳しい面もあるかもしれないが
ちょっとしたものを追加適用するだけで、十分やっていける。

■素のPHPでも、やり方次第で十分いけるだがやっておいた
方が後々幸せになれる。

■更に詳しい情報は下記URLを参照。

http://www.postgresql.jp/misc/seminar/2003-05-17/A3_ohgaki.pdf



（個人的に感じる）素のPHPでの厳しい面

大きなもので2点。

■1．スクリプト型言語である！

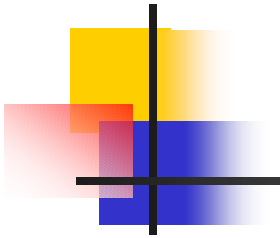
アクセスされる度にプログラムがコンパイルされ、実行
非常に長い行数の外部ファイルをインクルードしたり、大量の外部ファイルをインクルードした
場合に、コンパイル時の負荷は増大！更に、その状態で大量アクセスなんてされた日には。。。

■2．コネクションプーリングの機能がサポートされていない！

即ち「DBへの同時アクセス数 = Apacheのプロセス数」になってしまう！
ここでシステム規模が自動的に決まってしまう。

DBへの同時アクセス数が大きく取れないDBだと。。。

それらを解消する為に。。。



バイトコードキャッシュ・ソフトの紹介（有償無償含む）

■Zend Performance Suite

http://www.zend.com/store/products/zend-performance-suite.php

http://www.zend.co.jp/products/zps/

■ionCube PHP Accelerator

http://www.php-accelerator.co.uk/

http://www.ioncube.jp/

■Turck MMCache

http://www.turcksoft.com/en/e_mmc.htm

■afterBURNER*Cache

http://afterburner.bware.it/

■Alternative PHP Cache

http://pear.php.net/package/APC

■jpcache

http://www.jpcache.com/

■PHP MaxSpeed

http://www.pm9.com/newpm9/itbiz/php/

ざっと調べてみても、かなりのモノが発見できる。
これから言える事は、既にこの手法は一般的になりつつあるという事。

性能評価ーその1

今回はAPC Var2を使用してみての性能評価結果を行ってみた。

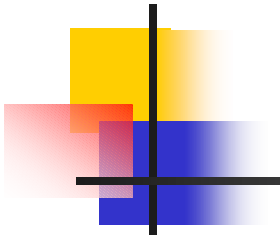
	メモリ使用率		総処理実時間		CPU占有時間	
hoge1.php	1986K	Byte	0.56	Sec	0.50	Sec
	66K	Byte	0.11	Sec	0.07	Sec
hoge2.php	2101K	Byte	0.72	Sec	0.61	Sec
	258K	Byte	0.26	Sec	0.21	Sec
hoge3.php	2002K	Byte	0.66	Sec	0.58	Sec
	155K	Byte	0.15	Sec	0.12	Sec
hoge4.php	1998K	Byte	0.60	Sec	0.49	Sec
	139K	Byte	0.15	Sec	0.12	Sec
hoge5.php	1994K	Byte	0.56	Sec	0.52	Sec
	113K	Byte	0.15	Sec	0.11	Sec
hoge6.php	2004K	Byte	0.59	Sec	0.53	Sec
	141K	Byte	0.17	Sec	0.13	Sec
hoge7.php	1985K	Byte	0.50	Sec	0.48	Sec
	88K	Byte	0.07	Sec	0.04	Sec

※　上はAPC導入前、下はAPC導入後

※　単発アクセスでの計測

この改善により、とあるシステムでは、APCを導入するだけで、PAC導入前と比べて10倍以上の処理を行う事が可能になった！本当にあった、怖い話2003年・夏。

これで、一つ幸せに。。。



コネクションプーリング・ソフトの紹介

- pgpool(個人的には今一番熱い！)

  http://www.sra.co.jp/people/t-ishii/PostgreSQL/index.html

- SQLB

  http://sqlb.sourceforge.net/

- SQL Relay

  http://sqlrelay.sourceforge.net/

何事も言ってみるもんだね。。。と思う今日この頃。

性能評価ーその2

今回はpgpoolを使用してみての性能評価結果を行ってみた。

計測ソフトはApache付属のab(-n200 -c1)。

	Time taken for testsの結果		Requests per secondの結果		処理成功件数・失敗件数
hoge1.php	18.851	Sec	10.61	Sec	199件・1件
hoge2.php	19.411	Sec	10.30	Sec	103件・97件
hoge3.php	23.276	Sec	8.59	Sec	200件・0件

これで、また一つ幸せになれそうだ。。。

特にpgpoolは、基本的にスクリプトの改修が

ほとんど必要ない！

さぁ、皆さんで人柱になりましょう。:-)

- Apache Max connect:32
- PostrgeSQL Max connect:16

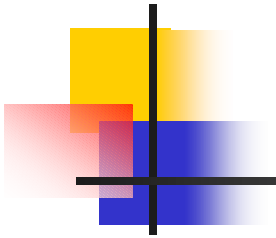
- hoge1.php:pg_connectを使用(通常接続で、PostgreSQLに直接接続)
- hoge2.php:pg_pconnectを使用(持続接続で、PostgreSQLに直接接続)
- hoge3.php:pg_connectを使用(通常接続で、pgpool経由でPostgreSQL接続)

- hoge1.phpはコネクト失敗時、1秒sleepさせてから再度コネクト処理を行う。その際、10回以上失敗時は強制終了。

- pgpoolの設定

  num_init_children = 4

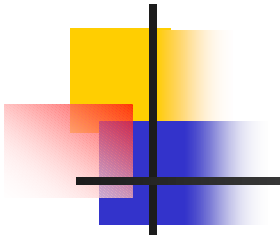
  max_pool = 1



勝手に 俺的結論

「WebアプリケーションにJavaは必要ない！」

色々な意味で勿体無い 、スクリプト言語で十分事足りる。
ただ、Javaがバックエンド的な要素でフロントエンドのPHPと連携するのはアリかも。



最後に

- パフォーマンステストは必ず行いましょう！
  目標数をもって行うのがベター。

- 特にCPU・メモリ利用率や、応答時間等は注意深くチェック！

- プログラム内部でのベンチマークは参考にならない場合があるので注意！

使えるものは、どんどん活用していきましょう。

ご清聴ありがとうございました。