通向架構師的道路（第三天）之apache性能調優 - lifetragedy的專欄 - 博客頻道

分類：

一、總結前一天的學習

在前兩天的學習中我們知道、瞭解並掌握了Web Server結合App Server實現單向Https的這樣的一個架構。這個架構是一個非常基礎的J2ee工程上線佈署時的一種架構。在前兩天的教程中，還講述了Http服務器、App Server的最基本安全配置（包括單向https的實現），它只是避免了用戶可以通過瀏覽器侵入我們的Web訪問器或者能夠通過Web瀏覽器來查詢我們的Web目錄結構及其目錄內的文件與相關內容，這種入侵我們把它稱為：

Directory traversal，當然我們只是實現了最基本的防範Directory traversal的手段，在日後的Security課程中將會詳細地去擅述完整的Web Security的相關理論。

從今天起我們將繼續在原有的這種Apache+Tomat的架構上，去論述如何在性能及Performance上優化這個架構，因此這兩天的課程在有些人看來，可能會有些「枯燥」，所以我在此給大家打個招呼：

這兩天的課程論述的是如何在不改動代碼與SQL語句的前提下，如何去改善和提高web server與app server的性能，千萬不要小覷這一內容，它可以讓你在不改動代碼的情況下得到10-20倍以上的性能提高，網上有其它的大牛們寫過一篇文章叫「Tomcat如何支持到1000個用戶」，經本人經過幾個重大工程的實踐，Opensource的Tomcat如果調優的好不只可以支持者1000個用戶，尤其當你的佈署環境是64位操作系統的情況下，可能能夠支持更大更高的並發性能，最後本節內容將會以Tomcat集群來做收場，在將來的課程中還會進一步詳細講述Weblogic的集群配置與IBM WASND的集群配置。

二、從性能測試談起

2.1 性能測試簡介

即壓力測試，就是根據一定數量的VU（Virtual Users）我稱為並發用戶操作核心交易後，系統所能達到的最大瓶勁，以便於發現系統的極限、有沒有Outof memory這樣的問題存在以及相關的系統設置、配置是否搭擋的合理的一種測試。

一般商業的比較好的用LoaderRunner，如果沒錢的就用Opensource的Jmeter來模擬這個VU的操作。

壓力測試，存在幾個誤區，需要小心。

1） 無限大的拚命增加VU的數量

系統再完美，硬件配置再高，也經不住沒有經過合理運算的VU的壓力呀。

2） 偏執的用一定的數據量的VU，跑7*24小時

不是說這個沒必要，很有必要，小日本的電視為什麼壽命敢說比中國人生產的電視機壽命長？因為它用一個機械臂就對著電視機的按鈕不斷的點點點。

我們說的壓力測試要測試多長時間，關鍵是要看經過科學計算的VU的數量以及核心交易數有多少，不是說我拿250個VU跑24*7如果沒有問題我這個系統就沒有問題了，這樣的說法是不對的，錯誤的。隨便舉個例子就能把你推倒。

假設我有250個VU，同時跑上萬筆交易，每個VU都有上萬筆交易，250個VU一次跑下來可能就要數個小時，你又怎麼能斷定250個VU對於這樣的系統我跑24*7小時就能真的達到上萬筆交易在250個VU的並發操作下能夠真的跑完7天的全部交易？可能需要一週半或者兩週呢？對吧？

我還看到過有人拿500個VU對著一條交易反覆跑24*7小時。。。這樣的測試有意義嗎？你系統就僅僅只有一條交易？你怎麼能夠判斷這條交易涉及到的數據量最大？更不用說交易是彼此間有依賴的，可能a+b+c+d的交易的一個混合組織就能夠超出你單筆交易所涉及到的數據量了呢！

2.2 合理的制定系統最大用戶、並發用戶

提供下面這個公式，以供大家在平時或者日常需要進行的性能測試中作為一個參考。

（1）　　計算平均的並發用戶數：C = nL/T

公式（1）中，C是平均的並發用戶數；n是login session的數量；L是login session的平均長度；T指考察的時間段長度。

（2）　　並發用戶數峰值：C' ≒ C+3根號C

公式（2）則給出了並發用戶數峰值的計算方式中，其中，C'指並發用戶數的峰值，C就是公式（1）中得到的平均的並發用戶數。該公式的得出是假設用戶的loginsession產生符合泊松分佈而估算得到的。

實例：

假設有一個OA系統，該系統有3000個用戶，平均每天大約有400個用戶要訪問該系統，對一個典型用戶來說，一天之內用戶從登錄到退出該系統的平均時間為4小時，在一天的時間內，用戶只在8小時內使用該系統。

則根據公式（1）和公式（2），可以得到：

C = 400*4/8 = 200

C'≒200+3*根號200 = 242

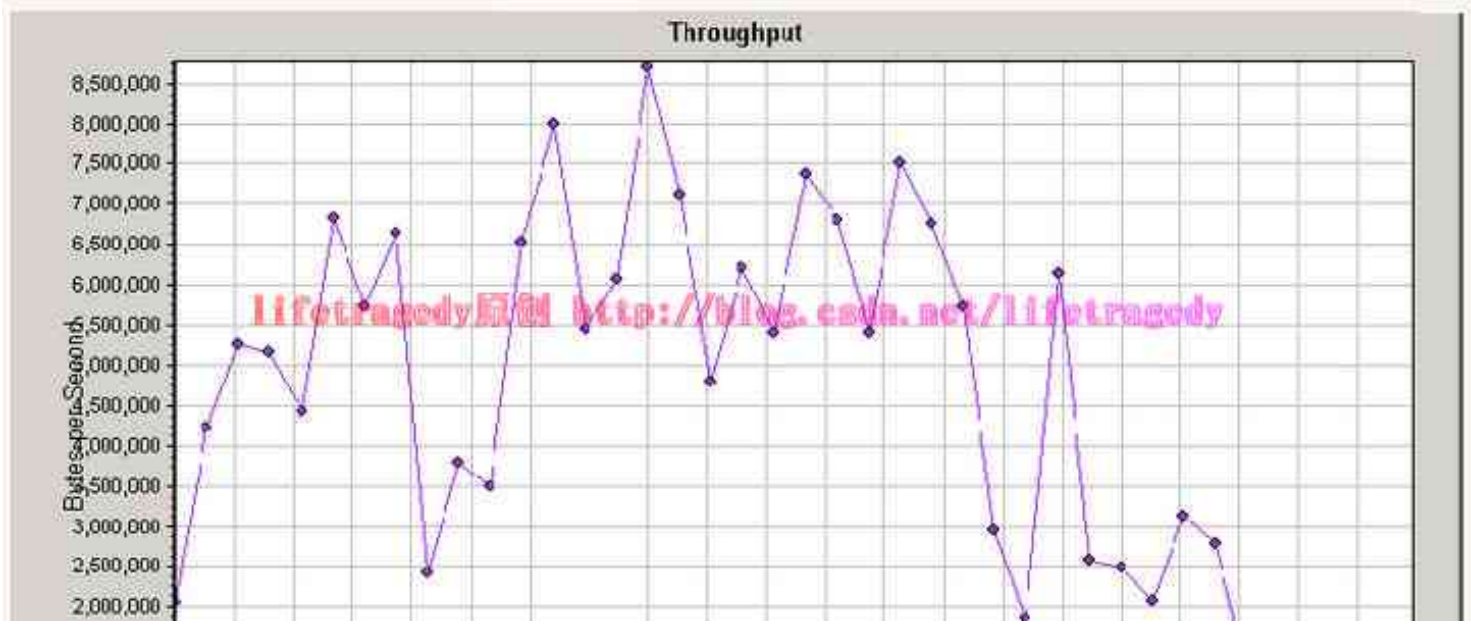
F=VU * R / T

其中F為吞吐量，VU表示虛擬用戶個數，R表示每個虛擬用戶發出的請求數，T表示性能測試所用的時間

R = T / TS。

2.3 影響和評估性能的幾個關鍵指標

從上面的公式一節中我們還得到了一個名詞「吐吞量」。和吞吐量相關的有下面這些概念，記錄下來以供參考。



² 吞吐量

指在一次性能測試過程中網絡上傳輸的數據量的總和。

對於交互式應用來說，吞吐量指標反映的是服務器承受的壓力，在容量規劃的測試中，吞吐量是一個重點關注的指標，因為它能夠說明系統級別的負載能力，另外，在性能調優過程中，吞吐量指標也有重要的價值。

² 吞吐率

單位時間內網絡上傳輸的數據量，也可以指單位時間內處理客戶請求數量。它是衡量網絡性能的重要指標，通常情況下，吞吐率用「字節數/秒」來衡量，當然，你可以用「請求數/秒」和「頁面數/秒」來衡量。其實，不管是一個請求還是一個頁面，它的本質都是在網絡上傳輸的數據，那麼來表示數據的單位就是字節數。

² 事務

就是用戶某一步或幾步操作的集合。不過，我們要保證它有一個完整意義。比如用戶對某一個頁面的一次請求，用戶對某系統的一次登錄，淘寶用戶對商品的一次確認支付過程。這些我們都可以看作一個事務。那麼如何衡量服務器對事務的處理能力。又引出一個概念----TPS

² TPS (Transaction Per second)

每秒鐘系統能夠處理事務或交易的數量，它是衡量系統處理能力的重要指標。

² 點擊率（Hit Per Second）

點擊率可以看做是TPS的一種特定情況。點擊率更能體現用戶端對服務器的壓力。TPS更能體現服務器對客戶請求的處理能力。

每秒鐘用戶向web服務器提交的HTTP請求數。這個指標是web 應用特有的一個指標；web應用是「請求-響應」模式，用戶發一個申請，服務器就要處理一次，所以點擊是web應用能夠處理的交易的最小單位。如果把每次點擊定義為一個交易，點擊率和TPS就是一個概念。容易看出，點擊率越大。對服務器的壓力也越大，點擊率只是一個性能參考指標，重要的是分析點擊時產生的影響。

需要注意的是，這裡的點擊不是指鼠標的一次「單擊」操作，因為一次「單擊」操作中，客戶端可能向服務器發現多

個HTTP請求。

² **吞吐量指標的作用：**

ü 用戶協助設計性能測試場景，以及衡量性能測試場景是否達到了預期的設計目標：在設計性能測試場景時，吞吐量可被用戶協助設計性能測試場景，根據估算的吞吐量數據，可以對應到測試場景的事務發生頻率，事務發生次數等；另外，在測試完成後，根據實際的吞吐量可以衡量測試是否達到了預期的目標。

ü 用於協助分析性能瓶頸：吞吐量的限制是性能瓶頸的一種重要表現形式，因此，有針對性地對吞吐量設計測試，可以協助盡快定位到性能冰晶所在位置。

² **平均相應時間**

也稱為系統響應時間，它一般指在指定數量的VU情況下，每筆交易從mouse 的click到IE的數據刷新與展示之間的間隔，比如說：250個VU下每筆交易的響應時間不超過2秒。

當然，響應時間也不能一概而論，對於實時交易如果銀行櫃檯操作、超市收銀員（邪惡的笑。。。）的操作、證交所交易員的操作來說這些操作的響應時間當然是越快越好，而對於一些企業級的如：

與銀行T+1交易間的數據跑批、延時交易、T+1報表等，你要求它在2秒內響應，它也做不到啊。就好比你有個1MB的帶寬，你傳的東西是超過4MB，你要它在2秒內跑完理論速度也做不到啊，對吧，所以有些報表或者數據，光前面傳輸時間就不止兩秒了。。。一口咬死說我所有的交易平均相應時間要2秒，真的是不科學的！

2.4 合理的性能測試

² **VU數量的增加**

一個合理的性能測試除了需要合理的計算VU的數量、合理的設置系統平均響應時間外還需要合理的在測試時去規劃發起VU的時間，比如說，我看到有人喜歡這樣做壓力測試。

第一秒時間，500個並發用戶全部發起了。。。結果導致系統沒多久就崩了，然後說系統沒有滿足設計要求。

為什麼說上述這樣的做法是不對的？我們說不是完全不對，只能說這樣的測試已經超過了500個VU的並發的設計指標了。

合理的並發應該是如下這樣的：

有25-50個VU開始起交易了，然後過一段時間又有25-50個用戶，過一段時間又增加一些VU，當所有的設計VU都發起交易了，此時，再讓壓力測試跑一段時間比如說：24*7是比較合理的。所以VU數量不是一上手就500個在一秒內發起的，VU數量的增加應該如下面這張趨勢圖：

這是一個階梯狀的梯型圖，可以看到VU的發起是逐漸逐漸增多的，以下兩種情況如果發生需要檢查你的系統是否在原有設置上存在問題：

ü VU數量上升階段時崩潰

有時僅僅在VU數量上升階段，系統就會了現各種各樣的錯誤，甚至有崩潰者，這時就有重新考慮你的系統是否有設置不合理的地方了。

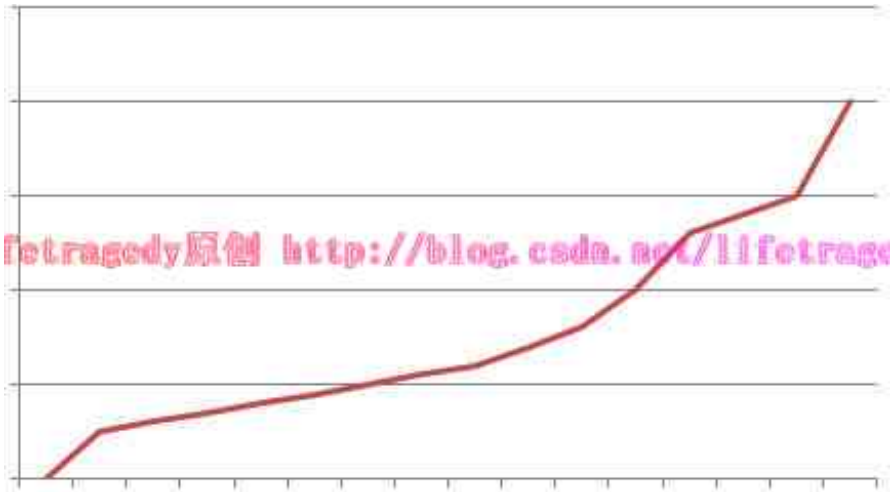
ü VU全部發起後沒多久系統崩潰

VU在達到最高值時即所有的VU都已經發起了，此時它是以一條直的水平線隨著系統運行而向前延伸著的，但過不了多久，比如說：運行24*7小時，運行了沒一、兩天，系統崩潰了，也需要做檢查。

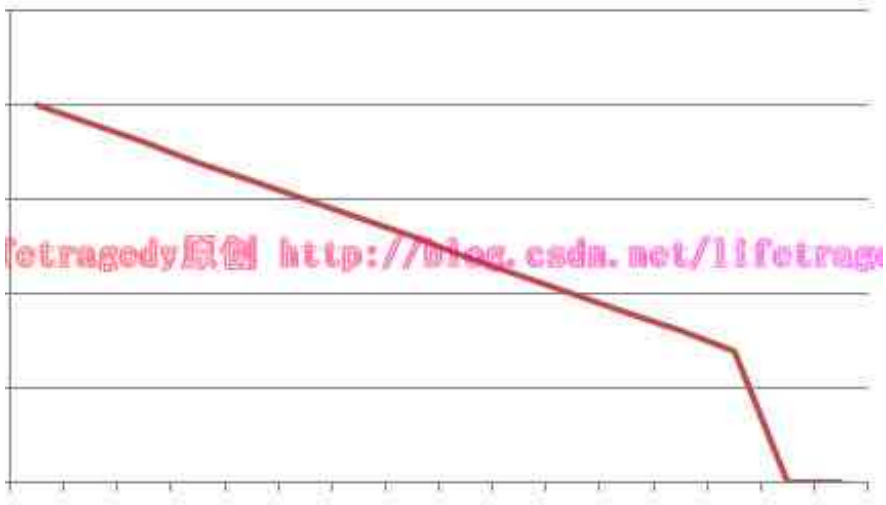
所以，理想的性能測試應該是VU數量上升到最終VU從發起開始到最後所有VU把交易做完後，VU數量落回零為止。

² 吐吞量的變化

從2.3節我們可以知道，吞吐量是隨著壓力/性能測試的時間而逐漸增大的，因此你的吞吐量指示應該如下圖所示：



肯定是這樣，你的吞吐量因該是積累的，如果你的吞吐量在上升了一段時間後突然下落，而此時你的性能測試還在跑著，如下圖所示：



那麼，此時代表什麼事情發生了？你可以查一下你的loaderrunner或者jmeter裡對於這段吞吐量回落期間的交易的response的狀態進行查看，你將會發現大量的error已經產生，因為產生了error，所以你的交易其實已經出錯了，因此每次運行的數據量越來越小，**這也就意味著你的壓力測試沒有過關，系統被你壓崩了！**

² **平均響應時間**

平均響應時間如VU的數量增加趨勢圖一樣，一定是一開始響應時間最短，然後一點點增高，當增高到一定的程度後，即所有的VU都發起交易時，你的響應時間應該維持在一個水平值，然後隨著VU將交易都一筆筆做完後，這個響應時間就會落下來，這段時間內的平均值就是你的系統平均響應時間。看看它，有沒有符合設計標準？

² **內存監控**

我們就來說AppServer，我們這邊用的是Tomcat即SUN的JVM的內存變化，我們就用兩張圖例來講解吧：

理想狀態情況下的JVM內存使用趨勢：

這是一個波浪型的（或者也可以說是鋸齒型的）趨勢圖，隨著VU數量的一點點增加，我們的內存使用數會不斷的增加，但是JVM的垃圾回收是自動回收機制的，因此如果你的JVM如上述樣的趨勢，內存上漲一段時間，隨即會一點點下落，然後再上漲一點，漲到快到頭了又開始下落，直到最後你的VU數量全部下降下來時，你的JVM的內存使用也會一點點的下降。

非理想狀態情況下的JVM內存使用趨勢：

嘿嘿嘿，看到了嗎？你的JVM隨著VU 數量的上升，而直線上升，然後到了一定的點後，即到了java –Xmx後的那個值後，突然直線回落，而此時你的交易還在進行，壓力測試也還在進行，可是內存突然回落了。。。因為你的JVM已經crash了，即OUT OF MEMORY鳥。

² CPU Load

我們來看一份測試人員提交上來CPU得用率的報告：

Web Server	App Server	DB Server
60%	98% =_=!(oh my god)	6% 囧

同時平均響應時間好慢啊。

拿過來看了一下代碼與設計。。。Struts+Spring+JDBC的一個框架，沒啥花頭的，再仔細一看Service層。

大量的複雜業務邏輯甚至報表的產生全部用的是javaobject如:List, Hashmap等操作，甚至還有在Service層進行排序、複雜查詢等操作。

一看DB層的CPU利用率才6%，將一部分最複雜的業務拿出去做成Store Procedure（存儲過程後），再重新運行壓力測試。

Web Server	App Server	DB Server
60%	57% =_=!(oh my god)	26% 囧

同時平均響應時間比原來快了15-16倍。

為什麼？？

看看第一份報告，我們當時還查看了數據庫服務器的配置，和APPServer的配置是一個級別的，而利用率才6%。。。

數據庫，至所以是大型的商用的關係型數據庫，你只拿它做一個存儲介質，你這不是浪費嗎？人家裡面設置的這個

StoreProcedure的處理能力，索引效率，數據分塊等功能都沒有去利用，而用你的代碼去實現那麼多複雜業務比如說多表關聯、嵌套等操作，用必要嗎？那要數據庫幹什麼用呢？

是啊，我承認，原有這樣的代碼，跨平台能力強一點，可付出的代價是什麼呢？

用戶在乎你所謂的跨平台的理論還是在乎的是你系統的效率？一個系統定好了用DB2或者是SQL SERVER，你覺得過一年它會換成ORACLE或者MySQL嗎？如果1年一換，那你做的系統也只能讓用戶勉強使用一年，我勸你還是不要去做了。在中國，有人統計過5年左右會有一次系統的更換，而一些銀行、保險、金融行業的系統一旦採用了哪個數據庫，除非這個系統徹底出了問題，負責是不會輕意換數據庫的，因此不要拿所謂的純JAVA代碼或者說我用的是Hibernate,ejb實現可以跨數據庫這套來說事，效率低下的系統可以否定你所做的一切，一切！

三、Apache服務器的優化

上面兩節，講了大量的理論與實際工作中碰到的相關案例，現在就來講一下在我們第一天和第二天中的ApacheHttp Server + Tomcat這樣的架構，怎麼來做優化吧。

3.1 Linux/UnixLinux系統下Apache 並發數的優化

Apache Http Server在剛安裝完後是沒有並發數的控制的，它採用一個默認的值，那麼**我們的Web Server硬件很好，允許我們撐到1000個並發即VU，而因為我們沒有去配置導致我們的WebServer連300個並發都撐不到**，你們認為，這是誰的責任？

Apache Http服務器採用prefork或者是worker兩種並發控制模式。

² preforkMPM

使用多個子進程，每個子進程只有一個線程。每個進程在某個確定的時間只能維持一個連接。在大多數平台上，PreforkMPM在效率上要比Worker MPM要高，但是內存使用大得多。prefork的無線程設計在某些情況下將比worker更有優勢：它可以使用那些沒有處理好線程安全的第三方模塊，並且對於那些線程調試困難的平台而言，它也更容易調試一些。

² workerMPM 使用多個子進程，每個子進程有多個線程。每個線程在某個確定的時間只能維持一個連接。通常來說，在一個高流量的HTTP服務器上，Worker MPM是個比較好的選擇，因為Worker MPM的內存使用比PreforkMPM要低得多。但worker MPM也由不完善的地方，如果一個線程崩潰，整個進程就會連同其所有線程一起"死掉".由於線程共享內存空間，所以一個程序在運行時必須被系統識別為"每個線程都是安全的"。

一般來說我們的ApacheHttp Server都是裝在Unix/Linux下的，而且是採用源碼編譯的方式來安裝的，我們能夠指定在編譯時Apache就採用哪種模式，為了明確我們目前的Apache採用的是哪種模式在工作，我們還可以使用httpd –l命令即在Apache的bin目錄下執行httpd –l，來確認我們使用的是哪種模式。

這邊，我們使用Apache配置語言中的」 IfModule」來自動選擇模式的配置。

我們的ApacheHttp Server在配完後一般是沒有這樣的配置的，是需要你手動的添加如下這樣的一塊內容的，我們來看，在httpd.conf文件中定位到最後一行LoadModule，敲入回車，加入如下內容：

```
<IfModule prefork.c>
ServerLimit  20000
StartServers  5
MinSpareServers  5
MaxSpareServers  10
MaxClients  1000
MaxRequestsPerChild 0
</IfModule>
```

上述參數解釋：

ü **ServerLimit 20000**

默認的MaxClient最大是256個線程,如果想設置更大的值，就的加上ServerLimit這個參數。20000是ServerLimit這個參數的最大值。如果需要更大，則必須編譯apache,此前都是不需要重新編譯Apache。

生效前提：必須放在其他指令的前面

ü StartServers 5

指定服務器啟動時建立的子進程數量，prefork默認為5。

ü MinSpareServers 5

指定空閒子進程的最小數量，默認為5。如果當前空閒子進程數少於MinSpareServers ，那麼Apache將以最大每秒一個的速度產生新的子進程。此參數不要設的太大。

ü MaxSpareServers 10

設置空閒子進程的最大數量，默認為10。如果當前有超過MaxSpareServers數量的空閒子進程，那麼父進程將殺死多餘的子進程。此參數不要設的太大。如果你將該指令的值設置為比MinSpareServers小，Apache將會自動將其修改成"MinSpareServers+1"。

ü MaxClients 256

限定同一時間客戶端最大接入請求的數量(單個進程並發線程數)，默認為256。任何超過MaxClients限制的請求都將進入等候隊列,一旦一個鏈接被釋放，隊列中的請求將得到服務。要增大這個值，你必須同時增大ServerLimit。

ü MaxRequestsPerChild10000

每個子進程在其生存期內允許伺服的最大請求數量，默認為10000.到達MaxRequestsPerChild的限制後，子進程將會結束。如果MaxRequestsPerChild為"0"，子進程將永遠不會結束。

將MaxRequestsPerChild設置成非零值有兩個好處：

1.可以防止(偶然的)內存洩漏無限進行，從而耗盡內存。

2.給進程一個有限壽命，從而有助於當服務器負載減輕的時候減少活動進程的數量。

Prefork.c的工作方式：

一個單獨的控制進程(父進程)負責產生子進程，這些子進程用於監聽請求並作出應答。Apache總是試圖保持一些備用的(spare)或者是空閒的子進程用於迎接即將到來的請求。這樣客戶端就不需要在得到服務前等候子進程的產生。在Unix系統中，父進程通常以root身份運行以便邦定80端口，而Apache產生的子進程通常以一個低特權的用戶運行。User和Group指令用於設置子進程的低特權用戶。運行子進程的用戶必須要對它所服務的內容有讀取的權限，但是對服務內容之外的其他資源必須擁有儘可能少的權限。

在上述的</IfModule>後再加入一個」<IfModule>」如下紅色加粗（**大又粗**）內容：

<IfModule prefork.c>

ServerLimit 20000

StartServers 5

MinSpareServers 5

MaxSpareServers 10

MaxClients 1000

MaxRequestsPerChild 0

</IfModule>

ServerLimit 50

ThreadLimit 200

StartServers 5

MaxClients 5000

MinSpareThreads 25

MaxSpareThreads 500

ThreadsPerChild 100

MaxRequestsPerChild 0

上述參數解釋：

ü ServerLimit16

服務器允許配置的進程數上限。這個指令和ThreadLimit結合使用設置了MaxClients最大允許配置的數值。任何在重啟期間對這個指令的改變都將被忽略，但對MaxClients的修改卻會生效。

ü ThreadLimit64

每個子進程可配置的線程數上限。這個指令設置了每個子進程可配置的線程數ThreadsPerChild上限。任何在重啟期間對這個指令的改變都將被忽略，但對ThreadsPerChild的修改卻會生效。默認值是"64".

ü StartServers3

服務器啟動時建立的子進程數，默認值是"3"。

ü MinSpareThreads75

最小空閒線程數,默認值是"75"。這個MPM將基於整個服務器監視空閒線程數。如果服務器中總的空閒線程數太少，子進程將產生新的空閒線程。

ü MaxSpareThreads250

設置最大空閒線程數。默認值是"250"。這個MPM將基於整個服務器監視空閒線程數。如果服務器中總的空閒線程數太多，子進程將殺死多餘的空閒線程。MaxSpareThreads的取值範圍是有限制的。Apache將按照如下限制自動修正你設置的值：worker要求其大於等於MinSpareThreads加上ThreadsPerChild的和

ü MaxClients400

允許同時伺服的最大接入請求數量(最大線程數量)。任何超過MaxClients限制的請求都將進入等候隊列。默認值是"400",16(ServerLimit)乘以25(ThreadsPerChild)的結果。因此要增加MaxClients的時候，你必須同時增加ServerLimit的值。

ü ThreadsPerChild25

每個子進程建立的常駐的執行線程數。默認值是25。子進程在啟動時建立這些線程後就不再建立新的線程了。

ü MaxRequestsPerChild 0

設置每個子進程在其生存期內允許伺服的最大請求數量。到達MaxRequestsPerChild的限制後，子進程將會結束。如果MaxRequestsPerChild為"0"，子進程將永遠不會結束。

將MaxRequestsPerChild設置成非零值有兩個好處：

1.可以防止(偶然的)內存洩漏無限進行，從而耗盡內存。

2.給進程一個有限壽命，從而有助於當服務器負載減輕的時候減少活動進程的數量。

注意

對於KeepAlive鏈接，只有第一個請求會被計數。事實上，它改變了每個子進程限制最大鏈接數量的行為。

Worker.c的工作方式：

每個進程可以擁有的線程數量是固定的。服務器會根據負載情況增加或減少進程數量。一個單獨的控制進程(父進程)負責子進程的建立。每個子進程可以建立ThreadsPerChild數量的服務線程和一個監聽線程，該監聽線程監聽接入請求並將其傳遞給服務線程處理和應答。Apache總是試圖維持一個備用(spare)或是空閒的服務線程池。這樣，客戶端無須等待新線程或新進程的建立即可得到處理。在Unix中，為了能夠綁定80端口，父進程一般都是以root身份啟動，隨後，Apache以較低權限的用戶建立子進程和線程。User和Group指令用於設置Apache子進程的權限。雖然子進程必須對其提供的內容擁有讀權限，但應該儘可能給予它較少的特權。另外，除非使用了suexec，否則，這些指令設置的權限將被CGI腳本所繼承。

公式：

ThreadLimit>= ThreadsPerChild

MaxClients <= ServerLimit * ThreadsPerChild 必須是ThreadsPerChild的倍數

MaxSpareThreads>= MinSpareThreads+ThreadsPerChild

硬限制：

ServerLimi和ThreadLimit這兩個指令決定了活動子進程數量和每個子進程中線程數量的硬限制。要想改變這個硬限制

必須完全停止服務器然後再啟動服務器(直接重啟是不行的)。

Apache在編譯ServerLimit時內部有一個硬性的限制，你不能超越這個限制。

preforkMPM最大為"ServerLimit200000"

其它MPM(包括work MPM)最大為"ServerLimit 20000

Apache在編譯ThreadLimit時內部有一個硬性的限制，你不能超越這個限制。

mpm_winnt是"ThreadLimit 15000"

其它MPM(包括work prefork)為"ThreadLimit 20000

注意

使用ServerLimit和ThreadLimit時要特別當心。如果將ServerLimit和ThreadLimit設置成一個高出實際需要許多的值，將會有過多的共享內存被分配。當設置成超過系統的處理能力，Apache可能無法啟動，或者係統將變得不穩定。

3.2 WindowsWindows系統下Apache 並發數的優化

以上是Linux/Unix下的Apache的並發數優化配置，如果我們打入了httpd –l如下顯示：

怎麼辦？

ü 步驟一

先修改/path/apache/conf/httpd.conf文件。

httpd.conf

將「#Includeconf/extra/httpd-mpm.conf」前面的 「#」 去掉，保存。

ü 步驟二

再修改/apache安裝目錄/conf/extra/httpd-mpm.conf文件。

在mpm_winnt模式下，Apache不使用prefork也不使用work工作模式，切記！

因此，我們只要找到原文件中：

```
<IfModule mpm_winnt_module>
    ThreadsPerChild      150
    MaxRequestsPerChild    0
</IfModule>
```

修改後

```
<IfModule mpm_winnt_module>
    ThreadsPerChild      500
    MaxRequestsPerChild    5000
</IfModule>
```

上述參數解釋：

ü ThreadsPerChild

是指一個進程最多擁有的線程數（Windows版本，貌似不可以開啟多個進程),一般100-500就可以，根據服務器的具體性能來決定。

ü MaxRequestsPerChild

是指一個線程最多可以接受的連接數，默認是0，就是不限制的意思，

0極有可能會導致內存洩露。所以，可以根據實際情況，配置一個比較大的值。Apache會在幾個線程之間進行輪詢，找到負載最輕的一個線程來接受新的連接。

注意：

修改後，一定不要apacherestart，而是先 apache stop 然後再 apache start才可以。

3.3 啟用服務端圖片壓縮

對於靜態的html 文件，在apache 可加載mod_deflate.so 模塊，把內容壓縮後輸出，可節約大量的傳輸帶寬。

打開httpd.conf文件，找到：

```
#LoadModule deflate_module modules/mod_deflate.so
```

將前面的「#」去掉，變成：

```
LoadModule deflate_module modules/mod_deflate.so
```

然後在最後一行的LoadModule處，加入如下的幾行：

```
<IfModule mod_deflate.c>
 DeflateCompressionLevel 7
 AddOutputFilterByType DEFLATE text/html text/plain text/xml application/x-httpd-php
 AddOutputFilter DEFLATE css js
</IfModule>
```

注意：

默認等級是6，而且9級需要更多的CPU時間，用默認的6級就可以了。

要注意的是,在apache 2.2.15中,我用httpd -l看,居然發現mod_deflat已經內置了，所以其實就不用再在httpd.conf中增加loadmodule了,否則會說出錯的

3.4 Apache中將MS辦公文檔自動關聯客戶端的MS-Office

我們經常會在web頁的一個超鏈接上點一個指向物理文件的文檔，我們一般會得到「保存，另存為，打開」，3個選項，當我們打開的如果是一個MS文檔，在選「打開」選項時IE會自動啟用客戶端上裝有的word或者是excel等相關MS辦公工具去打開，這個怎麼做呢？很簡單。

打開httpd.conf，找到：

```
AddType application/x-compress .Z
AddType application/x-gzip .gz .tgz
```

在其後敲入一個回車，加入：

```
AddType application/vnd.openxmlformats  docx pptx xlsx doc xls ppt txt
```

重啟Apache服務即可。

3.5 防止DDOS攻擊

DDOS攻擊即採用自動點擊機器人或者連續點擊工具不斷的刷新某一個網址或者網頁上的按鈕，造成網站在一時間收到大量的HTTP請求，進而阻塞網站正常的HTTP通道甚至造成網站癱瘓。

為了防止這一形式的攻擊，我們一般把在一個按鈕或者是一個請求在一秒內連續執行如：100次，可以認為是一種攻擊（比如說你打開一個網頁，點一下提交按鈕，然後按住F5鍵不松開）。

在Linux下的Apache HttpServer安裝後會提供一個mod_evasive20的模塊，用於防止這一形式的攻擊，它的做法是：

如果認為是一個DDOS攻擊，它的防範手段採用如下兩種形勢：

ü 把這個請求相關聯的IP，封鎖30分鐘

ü 直接把相關的IP踢入黑名單，讓其永不翻身

設置：

在你的Apache的httpd.conf文件中的最後一行「LoadModule」加入如下這句：

```
LoadModule evasive20_module   /usr/lib/httpd/modules/mod_evasive20.so
```

然後加入下面這幾行

```
<IfModule mod_evasive20.c>
DOSHashTableSize 3097
DOSPageCount 15
DOSSiteCount 100
DOSPageInterval 1
DOSSiteInterval 1
DOSBlockingPeriod 36000
DOSEmailNotify 網站超級管理員@xxx.com
DOSLogDir "logs/mod_evasive"
</IfModule>
```

核心參數解釋：

ü DOSHashTableSize3097 記錄黑名單的尺寸

ü DOSPageCount 每個頁面被判斷為dos攻擊的讀取次數

ü DOSSiteCount 每個站點被判斷為dos攻擊的讀取部件(object)的個數

ü DOSPageInterval 讀取頁面間隔秒

ü DOSSiteInterval 讀取站點間隔秒

ü DOSBlockingPeriod 被封時間間隔秒

注意：

上述設置是針對Linux/Unix下的Apache Server，相關的Windows下的Apache見如下設置：

為Windows下的Apache加載mod_evasive模塊

1. 下載附件中的壓縮包，解壓並拷貝mod_dosevasive22.dll到Apache安裝目錄下的modules目錄（當然也可以是其他目錄，需要自己修改路徑）。

2. 修改Apache的配置文件http.conf。

添加以下內容

```
LoadModule dosevasive22_module modules/mod_dosevasive22.dll
DOSHashTableSize 3097
DOSPageCount 3
DOSSiteCount 50
DOSPageInterval 1
DOSSiteInterval 1
DOSBlockingPeriod 10
```

3.6 Apache中設置URL含中文附件的下載/打開的方法（僅限Linux系統下）

這個話題很有趣，起因是我們在工程中碰到了客戶這樣的一個需求：

損壞部件

看看好像沒啥問題，一點這個超鏈接，因該是在IE中打開一個叫」 輪胎損壞情況2007-05-05.jpg」，嘿嘿，大家自己動手放一個帶有中文名的這樣的一個圖片，看看能否被解析，解析不了。

所以我們就說，真奇怪，我們上傳圖片都是上傳時的圖片名經上傳組件解析過以後變成一個UUID或者是GUID一類的文件名如：gb19070122abcxd.jpg這樣一種英文加數字組合的文件名，這樣的文件名，Apache當然是可以解析的，客戶堅持一定我上傳的圖片是中文名（連中文描述都不行），因為，客戶說：我們是中國人，當然用中文圖片名。。。

沒辦法，找了半天，找到一篇日文的教程，還好還好，N年前學過一點點日語，照著教程把它啃下來了。

這是一個日本人寫的關於在Apache中支持以亞州文字命名文件名的一個「補丁」，叫「mod_encoding」。

相關配置：

1. 下載完後是一個這樣的壓縮包：mod_encoding-20021209.tar.gz
2. 解壓後使用：

```
configure
make
make install
```

在make這一行時，編譯出錯，報「make: *** [mod_encoding.so] Error 1」這樣的錯

原因很明顯，是regex.h未包含進來，解決辦法也很簡單：

ü 用vi打開mod_encoding.c，

ü 在#include <httpd.h>那一段的前面加上如下一行：

#include <regex.h>然後：

重新make再make install 搞定，CALL！！！

3. 編譯後得到一個：mod_encoding.so的文件，然後在httpd.conf文件中加入下面這幾行：

```
LoadModule encoding_module modules/mod_encoding.so
Header add MS-Author-Via "DAV"
<IfModule mod_encoding.c>
  EncodingEngine    on
  NormalizeUsername on
  SetServerEncoding GBK
  DefaultClientEncoding UTF-8 GBK GB2312
  AddClientEncoding "(Microsoft .* DAV $)" UTF-8 GBK GB2312
  AddClientEncoding "Microsoft .* DAV" UTF-8 GBK GB2312
  AddClientEncoding "Microsoft-WebDAV*" UTF-8 GBK GB2312
</IfModule>
```

4. 重啟Apache，搞定，在apache中我們的url可以是中文名的附件了。

3.7 不可忽視的keepalive選項

在Apache 服務器中，KeepAlive是一個布爾值，On 代表打開，Off 代表關閉，這個指令在其他眾多的 HTTPD 服務器中都是存在的。

KeepAlive 配置指令決定當處理完用戶發起的 HTTP 請求後是否立即關閉 TCP 連接，如果 KeepAlive 設置為On，那麼用戶完成一次訪問後，不會立即斷開連接，如果還有請求，那麼會繼續在這一次 TCP 連接中完成，而不用重複建立新的 TCP 連接和關閉TCP 連接，可以提高用戶訪問速度。

那麼我們考慮3種情況：

1.用戶瀏覽一個網頁時，除了網頁本身外，還引用了多個javascript 文件，多個css 文件，多個圖片文件，並且這些文件都在同一個HTTP 服務器上。

2.用戶瀏覽一個網頁時，除了網頁本身外，還引用一個javascript 文件，一個圖片文件。

3.用戶瀏覽的是一個動態網頁，由程序即時生成內容，並且不引用其他內容。

對於上面3中情況，我認為：1 最適合打開 KeepAlive ，2 隨意，3 最適合關閉 KeepAlive

下面我來分析一下原因。

在 Apache 中，打開和關閉 KeepAlive 功能，服務器端會有什麼異同呢？

先看看理論分析。

打開KeepAlive 後，意味著每次用戶完成全部訪問後，都要保持一定時間後才關閉會關閉TCP 連接，那麼在關閉連接之前，必然會有一個Apache進程對應於該用戶而不能處理其他用戶，假設KeepAlive 的超時時間為10 秒種，服務器每秒處理 50個獨立用戶訪問，那麼系統中 Apache 的總進程數就是 10 * 50 = 500 個，如果一個進程佔用 4M 內存，那麼總共會消耗 2G內存，所以可以看出，在這種配置中，相當消耗內存，但好處是系統只處理了 50次 TCP 的握手和關閉操作。

如果關閉KeepAlive，如果還是每秒50個用戶訪問，如果用戶每次連續的請求數為3個，那麼 Apache 的總進程數就是 50 * 3= 150 個，如果還是每個進程佔用 4M 內存，那麼總的內存消耗為 600M，這種配置能節省大量內存，但是，系統處理了 150 次 TCP的握手和關閉的操作，因此又會多消耗一些 CPU 資源。

再看看實踐的觀察。

我在一組大量處理動態網頁內容的服務器中，起初打開KeepAlive功能，經常觀察到用戶訪問量大時Apache進程數也非常多，系統頻繁使用交換內存，系統不穩定，有時負載會出現較大波動。關閉了KeepAlive功能後，看到明顯的變化是：Apache 的進程數減少了，空閒內存增加了，用於文件系統Cache的內存也增加了，CPU的開銷增加了，但是服務更穩定了，系統負載也比較穩定，很少有負載大範圍波動的情況，負載有一定程度的降低；變化不明顯的是：訪問量較少的時候，系統平均負載沒有明顯變化。

總結一下：

在內存非常充足的服務器上，不管是否關閉KeepAlive 功能，服務器性能不會有明顯變化；

如果服務器內存較少，或者服務器有非常大量的文件系統訪問時，或者主要處理動態網頁服務，關閉KeepAlive 後可以節省很多內存，而節省出來的內存用於文件系統Cache，可以提高文件系統訪問的性能，並且系統會更加穩定。

ü 補充1

關於是否應該關閉 KeepAlive 選項，我覺得可以基於下面的一個公式來判斷。

在理想的網絡連接狀況下，系統的Apache 進程數和內存使用可以用如下公式表達：

HttpdProcessNumber= KeepAliveTimeout * TotalRequestPerSecond / Average(KeepAliveRequests)

HttpdUsedMemory= HttpdProcessNumber * MemoryPerHttpdProcess

換成中文意思：

總Apache進程數 = KeepAliveTimeout * 每秒種HTTP請求數 / 平均KeepAlive請求

Apache佔用內存 = 總Apache進程數 * 平均每進程佔用內存數

需要特別說明的是：

[平均KeepAlive請求] 數，是指每個用戶連接上服務器後，持續發出的 HTTP 請求數。當 KeepAliveTimeout 等 0或者 KeepAlive 關閉時，KeepAliveTimeout 不參與乘的運算從上面的公式看，如果 [每秒用戶請求]多，[KeepAliveTimeout] 的值大，[平均KeepAlive請求] 的值小，都會造成 [Apache進程數] 多和 [內存]多，但是當 [平均KeepAlive請求] 的值越大時，[Apache進程數] 和 [內存] 都是趨向於減少的。

基於上面的公式，我們就可以推算出當 平均KeepAlive請求 <= KeepAliveTimeout 時，關閉 KeepAlive 選項是划算的，否則就可以考慮打開。

ü 補充2

KeepAlive 該參數控制Apache是否允許在一個連接中有多個請求，默認打開。但對於大多數論壇類型站點來說，通常設置為off以關閉該支持。

ü 補充3

如果服務器前跑有應用squid服務，或者其它七層設備,KeepAlive On 設定要開啟持續長連接

實際在 前端有squid 的情況下,KeepAlive 很關鍵。記得On。

Keeyalive不能隨心所欲設置，而是需要根據實際情況，我們來看一個真實的在我工作中發生的搞笑一次事件：

當時我已經離開該項目了，該項目的TeamLeader看到了keepalive的概念，他只看到了關閉keeyalive可以節省web服務器的內存，當時我們的web服務器只有4gb內存，而並發請求的量很大，因此他就把這個keepalive設成了off。

然後直接導致脫機客戶端（脫機客戶端用的是.net然後webservice連接）的「login」每次都顯示「出錯」。

一查代碼才知道，由於這個脫機客戶端使用的是webservice訪問，.net開發團隊在login功能中設了一個超時，30秒，30秒timeout後就認為服務器沒有開啟，結果呢由於原來的apache設的是keeyalive和timeout 15秒，現在被改成了off，好傢伙，根本就沒有了這個timeout概念，因此每次.net登錄直接被apache彈回來，因為沒有了這個timeout的接口了。

由此可見，學東西。。。不能一知半解，務必求全面瞭解哈。

3.8 HostnameLookups設置為off

儘量較少DNS查詢的次數。如果你使用了任何」Allow fromdomain」或」Denyfrom domain」指令(也就是domain使用的是主機名而不是IP地址)，則代價是要進行兩次DNS查詢(一次正向和一次反向，以確認沒有作假)。所以，為了得到最高的性能，應該避免使用這些指令(**不用域名而用IP地址也是可以的**)。