規模に応じたインターネットサーバ構築運用ノウハウ

民田 雅人

（松井証券（株））

1999 年 12 月 16 日

Internet Week 99 パシフィコ横浜

（社）日本ネットワークインフォメーションセンター編

目次

1　概要

世の中のインターネットサーバには、ISP や新聞社などの WEB サイトで使われているようなアクセスが多く高負荷で大規模なものから、アクセスが少なく低負荷で小規模のものまでさまざまなものがあります。適正なマシンを適正なコストで運用することが重要で、これを正確に理解しておくことが、システムが大規模になるほどコストがかかるという問題への対応策を考える際に大切になります。私の運用経験から、この講演では、Solaris を使用したサーバと FreeBSD を使用したサーバについてのノウハウを紹介します。

2　システム構成

2.1　規模とコスト

概して、大規模なシステムを構築すると初期コストが高くつきます。一方で、小規模なシステムではハードウェアの初期コストは安く、構築には手間もかかりません。しかし、負荷が上がってくるとシステムの運用にノウハウが必要になります。大きなシステムを小さなコストで運用できれば理想ですが、なかなかそうはいかず、大規模になると運用コストは高くなってしまいます。

マシン性能が速いこととコストとの関係ですが、速いことは、いろいろと良いことです。CPU が速いと計算処理が短時間になります。バスが速いと全体のスループットが速くなります。ハードディスクが速ければデータの読み書きが短時間になり、全体の処理が速くなります。アプリケーションの起動も速くなり、スワップのスピードも速くなります。ネットワークの回線容量が大きければデータ転送が短時間で終わります。

スピードとコストの関係を初期コストと運用コストとに分けて考えてみましょう。

速いマシンを使用したシステムは、もともと処理能力が高いために手間がかからず、人件費が少なくて済むだけでなく、後々のメンテナンスも楽になり運用コストが安くなります。

遅いマシンを使用したシステムは、処理能力を上げるための手間がかかるため、2、3 年間で見た運用コストと合わせたトータルのコストでは、かえって高くつき後悔することがよくあります。ですから、このあたりをきちんと見極める必要があります。

2.2 インターネットサーバの種類

インターネットサーバにもさまざまな種類があります。

インターネットサーバの代表的なものは、WWW サーバ、ネームサーバ、メールサーバ、NEWS サーバ、WEB のキャッシュサーバ、IRC サーバなどがあります。

インターネットの代名詞といわれる WWW サーバ、インターネットの運用に重要なドメイン名から IP アドレスへとマッピングしたり、IP アドレスからドメイン名へとマッピングするネームサーバ、およびメールサーバの 3 種類が最もよく知られているサーバです。

そのほかには、最近あまりにも高トラフィックで大変になっている NEWS サーバ、技術的に面白い WEB キャッシュサーバ、IRC サーバなどがあります。また、ストリーミングでデータを送るサーバなどもあります。

インターネットサーバを作る場合には、CPU の種類とスピード、メモリの容量、ハードディスクの台数と容量、ネットワークインタフェースのスピード（10Mbps か 100Mbps）などの構成要素の組み合わせが性能を決める重要なポイントになります。

2.3 CPU 選択のポイント

CPU を選択する場合は、アーキテクチャ、クロック周波数、キャッシュのスピードと容量、メモリバスなどがポイントです。

2.3.1 CPU のアーキテクチャ

CPU のアーキテクチャといってもいろいろあります。主な CPU アーキテクチャとしては、PC/AT 互換機の x86（i386)、Sun とその互換機の SPARC、IBM や Apple Macintosh の PowerPC、SGI の MIPS、COMPAQ の Alpha、HP と日立製作所の PA-RISC などがあります。

この中で、一番数が多いのが PC/AT 互換機の x86 アーキテクチャです。PC/AT 互換機に FreeBSD や Linux を搭載し、インターネットサーバとして使っているケースは珍しいものではありません。また、SPARC もよく使われています。このほかのアーキテクチャも使われていますが、PowerPC 搭載の Macintosh はインターネットのサーバとしてはほとんど使われていません。

2.3.2 PC/AT 互換機の CPU の変遷

PC/AT 互換機、i386 アーキテクチャの CPU クロックを示します。

●i486	25～100MHz
●Pentium	75～200MHz
●MMX テクノロジ Pentium	166～266MHz

●Pentium Pro　　150～200MHz

これらは、すでに過去の CPU と言えるものですが、性能は CPU ベンチマークの現行指標である SPECint95 に換算すると 0.3 から 8 に相当します。

次に現行の CPU を示します。

●PentiumⅡ　　233～450MHz
●PentiumⅡ Xeon　400～450MHz
●PentiumⅢ　　450～733MHz
●AMD-K6-Ⅲ　　350MHz～
●AMD　Athlon　　500～700MHz

SPECint95 で 9.5 から 22 と、最初の i486 CPU の性能に比べると PC/AT 互換機の CPU の性能は 100 倍に向上しています。

2.3.3　SPARC の変遷

次に SPARC の変遷として、過去に登場した種類を示します。

●MicroSPARC（LX，Classic）　　50MHz
●MicroSPARC-Ⅱ　70～110MHz（SPARC Station5 に搭載）
●TurboSPARC　　170MHz（SPARC Station5 に搭載）
●SuperSPARC
●HyperSPARC

一方、以下が現行の CPU 製品です。

●UltraSPARC　　140MHz/167MHz（Ultra1 に搭載）
●UltraSPARC-Ⅱi　270～480MHz（Ultra5、10 に搭載）
●UltraSPARC-Ⅱ　250～450MHz（Enterprise450 など）

これらは、SPECint95 で 6.6 から 20 程度です。

2.3.4　CPU クロックスピード

CPU 選択の指標として一番わかりやすいのが、CPU クロックのスピードです。同じアーキテクチャの CPU だと、クロック数が大きいほど CPU は速くなります。処理能力とクロックとは密接な相関関係があります。ただ、同じクロック数でもアーキテクチャの違いや周辺回路などの設計が異なると、必ずしも単純に比例しないことがあります。

CPU の性能を比較する具体例として、FreeBSD カーネルをコンパイルした時間を紹介します。これは、私が記録している例です。

手元にある 466MHz 版 Celeron が最も速くて、2 分以下でコンパイルが終了します。最近になると FreeBSD カーネルのバージョンが 3.3 と新しく、カーネルが大きくなっているので単純比較はできませんが、333MHz 版 PentiumⅡで 2 分程度、150MHz 版 Pentium Pro で 4 分か

ら 5 分程度、120MHz 版 Pentium で 10 分程度、66MHz 版 486DX2 ではカーネルのバージョンが 2 以前で、またハードディスクも遅かったので 20 分から 40 分程度かかりました。構成が異なるので一概に比較できませんが、実際にはこの数字以上の差があるといって良いでしょう。

2.3.5 キャッシュメモリ

いまどき、キャッシュなしにまともに稼働できる CPU はありません。CPU がどんどん速くなるのに対してメモリはその速度に追い付いていけないため、キャッシュが CPU とメモリの速度の差を吸収しています。

私は、CPU のカタログを見るときに必ずキャッシュの容量や動作などの仕様に注意しています。

キャッシュには 1 次キャッシュと 2 次キャッシュがあり、ものによっては 3 次キャッシュというのもあります。

1 次キャッシュは CPU に内蔵されています。4KB から、多いものでは AMD Athlon のように 128KB の容量のものがあります。

2 次キャッシュは一般に CPU の外部にありますが、x86 の CPU では Pentium Pro 以降で内蔵されています。2 次キャッシュの容量は、128KB から、多いものでは SPARC の場合で 8MB というものもあります。

キャッシュの能力を判断するためには、まずキャッシュサイズに注目します。ただ、それだけでは十分ではなく、キャッシュのメモリバススピードが重要になります。

注意しなければいけないのが、キャッシュが可能なエリアがあることです。現行の SPARC 製品や x86 系製品を使用する場合にはあまり気にしなくてもかまいませんが、旧製品を使用するときには、キャッシュが可能なエリアの制約が問題になります。

たとえば、333MHz 版 PentiumⅡまでの CPU のケースです。PentiumⅡの 333MHz 版までは、キャッシュ可能なエリアが 512MB でした。この場合、主記憶メモリを 512MB より多く、たとえば 1GB をマザーボードに搭載しても、それはメモリの積み過ぎということになります。キャッシュが効いていないので、思ったほど性能が上がらないことになります。現在の PentiumⅢや Celeron は 4GB までキャッシュ可能なエリアが拡大したので、あまり問題にしなくてもよいかもしれません。

x86 系のキャッシュサイズは、Pentium Pro の 1 次キャッシュが合計 16KB（プログラムコード用 8KB、データ用 8KB）、2 次キャッシュが 256KB～1MB です。450MHz 以上の PentiumⅢが 1 次キャッシュ 32KB（コード用 16KB、データ用 16KB）、2 次キャッシュが 512KB です。最近の PentiumⅢはカッパーマインと呼ばれるもので、2 次キャッシュは 256KB と半分になっています。

AMD-K6-2 は、1 次キャッシュがインテルのものよりも大きくて 64KB（プログラムコード用 32KB、データ用 32KB）となっています。実は、この 1 次キャッシュが大きいことがおそらく AMD の CPU が速い理由

のひとつです。2 次キャッシュは外付けで、マザーボードの構成によって異なっています。ちなみに、AMD Athlon の 1 次キャッシュは 128KB です。

次に、SPARC のキャッシュサイズを示していきます。

Ultra5 や Ultra10 で使われている現行の UltraSPARC-IIi は、1 次キャッシュが 32KB（コード用 16KB、データ用 16KB）あり、2 次キャッシュが 270MHz 版では 256KB、300MHz 版では 512KB、333MHz 版では 2MB あります。最近では、480MHz 版で 4MB のキャッシュを持つものもあります。

Sun の大型マシンの多くで搭載されている UltraSPARC-IIは、1 次キャッシュが 32KB（コード用 16KB、データ用 16KB)、2 次キャッシュが 250MHz 版では 1MB、300MHz 版では 2MB あります。400MHz 版では、4MB と 8MB のものとの 2 種類があります。

当然、2 次キャッシュを多く搭載している CPU は価格が高くなります。

キャシュメモリのバススピードと CPU クロックがシステムの処理性能にどう影響するかの一例ですが、以前の比較例で PentiumII と Pentium Pro を比べた面白い話があります。

通常は PentiumIIが速いのですが、233MHz から 333MHz までの PentiumIIの 2 次キャッシュは CPU クロックスピードの半分のバススピードで動作します。したがって、たとえば 300MHz の PentiumIIであっても 2 次キャッシュのバススピードは 150MHz となるわけです。しかも、キャッシュ可能なエリアは 512MB しかありません。

これに対し Pentium Pro は、CPU クロックは PentiumIIよりも遅いのですが、たとえば 200MHz 版 Pentium Pro では２次キャッシュのバススピードは CPU クロックと同じで 200MHz になります。キャッシュ可能エリアは 4GB まであります。したがって、Pentium Pro の 200MHz 版は、キャッシュにヒットする限り CPU のクロックスピードが生かされて速いスピードで命令やデータの読み込みが可能になります。非常に巨大なデータを扱うようなケースでは、333MHz 版 PentiumIIの場合には 2 次キャッシュのバススピードがネックになり、Pentium Pro のほうが速い処理になることがあります。

次に Celeron と PentiumIIIの性能を比較してみましょう。ここでは、Celeron、PentiumIII、PentiumIII（E）を比較しています。

表 1：Celeron と PentiumIIIのキャッシュ性能

	Celeron	PentiumIII	PentiumIII(E)
1次Cache	16K + 16K	16K + 16K	16K + 16K
2次Cache	128K	512K	256K
Cache clock	CPU	½ CPU	CPU
FSB	66M	100 or 133M	100 or 133M
CPU Clock	～500MHz	～600MHz	～733MHz
お値段	安い	高い	かなり高い

1 次キャッシュの容量は同じです。2 次キャッシュは PentiumIIIが多くなっています。PentiumIII（E）では、2 次キャッシュは PentiumIIIの半分となっています。1 次キャッシュのクロックを見ると、Celeron、PentiumIII（E）が CPU クロックと同じなのに対して、PentiumIIIでは CPU クロックの半分になっています。FSB（フロントサイドバス）は 2 次キャッシュのバススピードです。Celeron が 66MHz なのに対して、PentiumIII、PentiumIII（E）は 100MHz または 133MHz となっています。

ところで、実勢市場価格ですが、Celeron は 1 万 5000 円程度、Pentium IIIは 4 万 5000 円程度、PentiumIII（E）は 8 万円を超えています。値段の割に Celeron は性能が良いと言えるでしょう。

2.3.6 メモリバス

メモリバスは CPU とメモリのインタフェースのことで、メモリバスのクロックとバス幅で転送速度が決まります。現在のメモリバスのバス幅には、32bit、64bit、128bit などがあります。

理論値だけで計算しますと、バス幅が 32bit でクロックが 33MHz のメモリバスでは 132MByte/sec のデータ転送ができます。

バス幅 64bit でクロック 66MHz のメモリバスでは 528MByte/sec、バス幅 64bit でクロック 100MHz のメモリバスでは 800MByte/sec、128bit でクロックが 100MHz の場合は 1600MByte/sec のデータが転送できます。しかし、これは単純な理論値で、しかもデータを連続的に転送する条件でのことで、実際には、計算通りの転送レートは得られません。

実際のメモリの動作には、最初のアクセスに何クロックかかって続くデータの転送に何クロックかかるかというメモリのアクセスパターンと呼ばれるものが影響します。これは、マザーボード、チップセット、メモリの種類によっていろいろ変化します。

たとえば、最初の 1 ワードに 4 クロック、それに続く 1 ワード当たり 2

クロックかかるアクセスパターンの場合、4 ワードを転送するのに 4－2－2－2 となり、トータルで 10 クロックになります。

ところが、最初の 1 ワードに 5 クロックかかっても、それに続く 1 ワードごとに 1 クロックしかかからない 5－1－1－1 のアクセスパターンの場合には、トータル 4 ワードを 8 クロックで転送できることになります。

連続アクセスならば後者のほうが転送レートは速くなりますが、ランダムアクセスとなると最初の 1 ワードのアクセスの部分が増えるわけですから後者のほうが遅くなることもあります。

簡単なコードを書いて、いろいろな CPU におけるメモリの実際の転送速度を計測してみました。C の標準関数 memcpy を次のように繰り返し実行させています。

```
for(i=0;i<=n;i++)memcpy(dst,src,len);
```

時間計測の誤差を考慮して、トータルで 1GB となるサイズになるように 10MB を 100 回コピーし、それに要する時間を計測しました。その際にマシンに memcpy の for 文だけを実行させて、「gettimeofday」で開始時間と終了時間をマイクロ秒単位で計測し、引き算で値を求めました。最近の UNIX はマイクロ秒単位までの計時精度があります。それに対し、Windows ではそれほどの精度はなく、こういった計測には向いていません。

計測結果ですが、300MHz 版 UltraSPARC-Ⅱが 281MByte/sec でした。400MHz 版 UltraSPARC-Ⅱが 276MByte/sec、167MHz 版 UltraSPARC-Ⅱが 170MByte/sec、400MHz 版 PentiumⅡが 140MByte/sec、300MHz 版 UltraSPARC-Ⅱi が 132MByte/sec でした。466MHz 版 Celeron が 86MByte/sec、466MHz 版 Celeron を別のマシンに搭載したケースが 45MByte/sec で、50MHz 版 SuperSPARC が 29MByte/sec でした。

300MHz 版 UltraSPARC-Ⅱと 400MHz 版 UltraSPARC-Ⅱは同じマシンで計測した結果ですが、CPU クロックが速くなってもメモリバススピードは同じで、メモリ間で受け渡す限りは CPU クロックが速くなってもほとんど差はでないという典型を示しています。

概して、SPARC のバスはメモリの転送が非常に速くできると言えます。

現在最も速いのは 300MHz 版 UltraSPARC-Ⅱと 400MHz 版 UltraSPARC-Ⅱのクラスですが、このスピードは実際には主記憶上でデータをコピーしているのですが、Pentium Pro の 2 次キャッシュ内でコピーしているのとほとんど変わらないスピードです。これはメモリバスのバンド幅が 512bit という非常に幅の広いバスを採用しているためです。

2.3.7 メモリ容量

システムを構築するには必要にして十分なメモリ容量を用意するというのが基本です。少ないとアプリケーションすべてが稼働できない事態も生じます。その際にスワップが起きますが、たとえ稼働できてもパフォ

ーマンスが落ちます。多過ぎるメモリというのは単に無駄で、コストに影響します。

メモリ容量選択のポイントですが、サーバで稼働するアプリケーションがどの程度メモリを消費するか、OS および OS が標準で動かす daemon 類を含めてどのくらいメモリがないと動かないか、現在の OS は余ったメモリ容量をディスクバッファに利用するので、これらを見極めてメモリの実装量を計算する必要があります。

2.3.8 SPARC vs. x86

ところで、SPARC と x86 のいずれが速いかですが、処理する内容によって異なってきます。一般には整数演算は x86 が速く、浮動小数点演算は SPARC が速いと言われています。しかし、実際には思わぬものが速かったりしますので、選ぶときには同じ条件で実際にテストしてみないとわからないというのが現実でしょう。

実際に行ったベンチマークの例を図 1 と図 2 で示します。図 1 は、ssh-keygen というセキュアシェルのデータを暗号化して遠隔地のマシンと通信を行う ssh のキーを生成するプログラムを使って、生成時間の簡単なベンチマークを行った結果です。

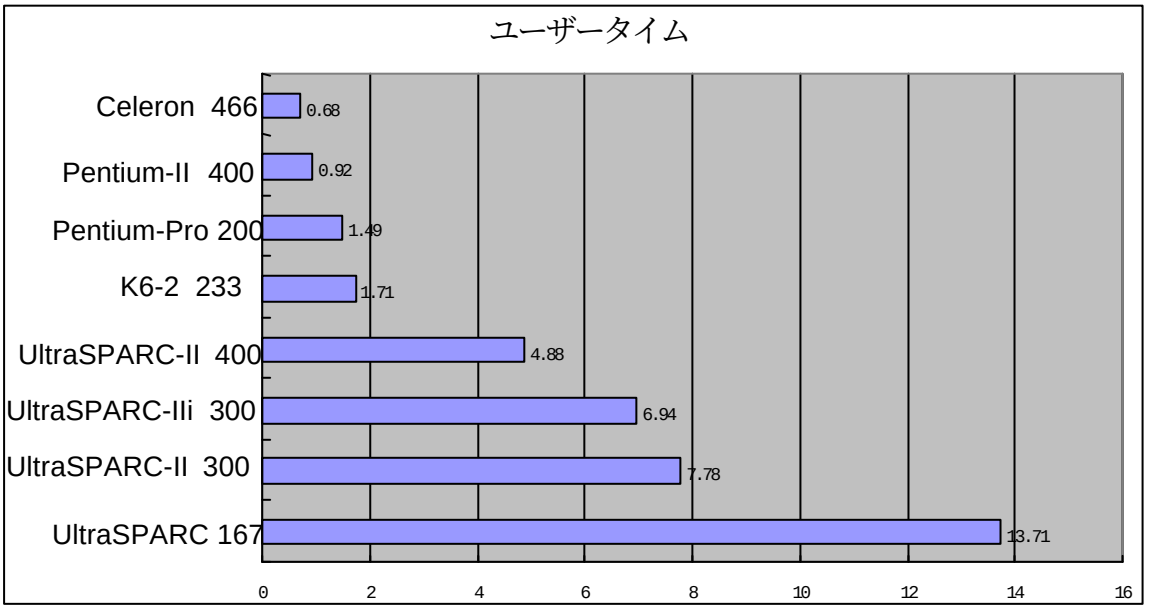

図 1：ssh-keygen のベンチマーク

これで最も速かったのは 466MHz 版 Celeron で、UltraSPARC-IIはいずれも x86 より時間がかかるという結果が見られました。

また、整数演算の gzip でのベンチマークを行ってみました。それが次

の図 2 です。135MB のファイルの gzip にかかるユーザタイム（秒）を計測したものです。

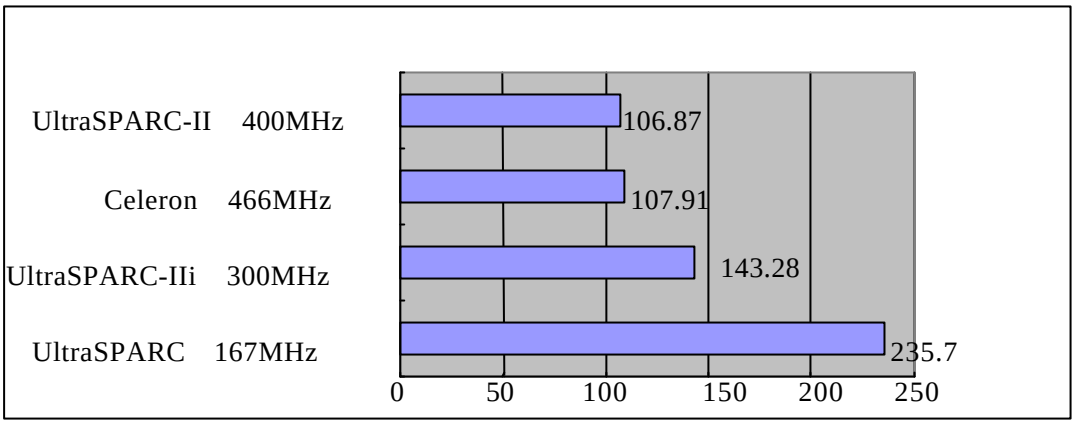

図 2：gzip のベンチマーク

このベンチマークでは、UltraSPARC-IIの方が速いという結果が出ました。

概してデータ転送は SPARC が速いのは確かですが、それ以外については、実際は条件によって変わるということです。

2.4 ハードディスク

ハードディスクは、ヘッドが回転する磁気円盤上に磁気情報を書き込み、読み出しを行っているものです。磁気円盤に同心円状に記憶されます。同一円上にあるトラックを対象にした連続アクセスの場合には、ヘッドは同じ位置で読み取りや書き込みを行いますから比較的速いデータ転送が可能です。しかし、ランダムアクセス時にはヘッドが移動するために、その移動時間がアクセスを遅くする原因になります。そこで、ランダムアクセスをいかに減らすかの工夫が課題になってきます。

デスクトップでは IDE も使われていますが、インターネットサーバでは一般的に SCSI インタフェースが使われています。

HDD のカタログ情報を見るときに注意する点を挙げます。

例として実際に IBM の DDRS シリーズ（34560/4.3GB と 39130/9GB）のカタログスペックを示します。

●Media data rate	109〜171Mbps
●Rotational speed	7200rpm
●Sustained data rate	8.3〜13.3 MByte/s
●Average seek time	7.5ms
●Average latency	4.33ms

Media data rate は、ハードディスクの円盤上にある生のデータを連続的に読み出す場合のスピードで、実際には磁気記録密度で決まります。

Rotational speed は円盤の回転スピードで、現在は 1 分間に 1 万回転するものまであります。

Sustained data rate は、普通に連続読み出し/書き込みを行うときのスピードです。

Average seek time は、ヘッドが移動して次のデータの読み出しを行うまでの平均的な待ち時間のことです。

Average latency は、実際にデータが転送されてくるまでの時間です。

2.5 SCSI コントローラ

ハードディスクを接続するのが SCSI コントローラです。一般的な SCSI コントローラは、DMA（Direct Memory Access）方式によって CPU を介さずにデータ転送を行います。CPU がコントローラに対してコマンドを送ると、あとはコントローラがメモリとディスク間のデータ転送を行ってくれます。CPU はコマンドを送ったあとは次の作業に進めます。

しかし、注意しなければならないのは、ハードディスクに対して書き込み（WRITE）を行う場合と読み出し（READ）を行う場合で CPU の動作は異なってくることです。

CPU は、書き込みの場合と違って、読み出しの場合には読み込んだ結果（データ）が現在稼働させているアプリケーションの実行に必要なわけですから、どうしてもディスク側の読み出し動作の完了を待たなければならず、読み出しの方が CPU の待ち時間が増え、パフォーマンスにも影響を与えることになります。これはネットワークの場合にも起こります。

複数の SCSI コントローラを付けることにより SCSI バスを複数にすることができ、バスに接続したデバイスに同時に書き込むことができるようになります。ハードディスクの負荷分散にこの方法がよく使われます。

2.6 ネットワークインタフェース

サーバのネットワークインタフェースは、10BaseT や 100BaseTX を使うケースがほとんどですが、最近では特に 100BaseTX を使用するのが常識になっており、ほとんど選択の余地がありません。また、スイッチと組み合わせて full-duplex にする際には品質には注意する必要があります。100Mbps のインタフェースで FDDI がありますが、価格がどうしても高く、あまり使われていません。

最近の CPU は相当に速くなっていますから、100Mbps の帯域全部を使いきるケースもあり、ネットワーク周りへの配慮も必要になってくるでしょう。

3 Operating System

3.1 OS のチューニング

私は「OS を与えられたまま使っていては負けだ」と常々言っています。Solaris を標準でインストールすると、実際に使わないものまで実にさまざまな daemon が動き出します。使わないものは止めるのが正解です。

変えられるカーネルなら変えられるパラメータは変えてしまいます。プロセスを止めることによってメモリを節約できます。プロセステーブルから不要なプロセスが減ることによってシステムのスループットが向上します。

Solaris2.6 を標準インストールした場合の「ps –efa」の結果を以下に示します。

Solaris 2.6 の標準状態(チューニング前)

ps -efa の結果

```
UID    PID   PPID C   STIME TTY      TIME CMD
root     0      0  0   0ct 13 ?      0:01 sched
root     1      0  0   0ct 13 ?      0:00 /etc/init -
root     2      0  0   0ct 13 ?      0:00 pageout
root     3      0  0   0ct 13 ?     13:12 fsflush
root   195      1  0   0ct 13 ?      0:00 /usr/lib/sendmail -bd -q1h
root   167      1  0   0ct 13 ?      0:03 /usr/sbin/cron
root   277      1  0   0ct 13 ?      0:00 /usr/lib/saf/sac -t 300
root    75      1  0   0ct 13 ?      0:00 /usr/sbin/aspppd -d 1
root    96      1  0   0ct 13 ?      0:00 /usr/sbin/in.rdisc -s
root   106      1  0   0ct 13 ?      0:00 /usr/sbin/rpcbind
root   133      1  0   0ct 13 ?      0:00 /usr/sbin/inetd -s
root   108      1  0   0ct 13 ?      0:00 /usr/sbin/keyserv
root   153      1  0   0ct 13 ?      0:00 /usr/sbin/syslogd -n -z 12
root   138      1  0   0ct 13 ?      0:00 /usr/lib/nfs/statd
root   140      1  0   0ct 13 ?      0:00 /usr/lib/nfs/lockd
root   173      1  0   0ct 13 ?      0:00 /usr/sbin/nscd
root   183      1  0   0ct 13 ?      0:00 /usr/lib/lpsched
root   280      1  0   0ct 13 ?      0:00 /usr/dt/bin/dtlogin -daemo
root   212      1  0   0ct 13 ?      0:02 /usr/sbin/vold
root   218    216  0   0ct 13 ?      0:00 /usr/sbin/ccv -f
root   205      1  0   0ct 13 ?      0:00 /usr/lib/utmpd
root   216      1  0   0ct 13 ?      0:00 /usr/sbin/cssd
root   217    216  0   0ct 13 ?      0:00 /usr/sbin/cs00
root   219    216  0   0ct 13 ?      0:00 /usr/sbin/kkcv -f
root   221      1  0   0ct 13 ?      0:00 /usr/lib/locale/ja/wnn
                                          /dpkeyserv
root   225      1  0   0ct 13 ?      0:00 /usr/lib/locale/ja/wnn
                                          /jserver
```

```
root   226   225  0   Oct 13 ?        0:01 /usr/lib/locale/ja/wnn
                                          /jserver_m
root   281   277  0   Oct 13 ?        0:00 /usr/lib/saf/ttymon
root   278     1  0   Oct 13 console  0:00 /usr/lib/saf/ttymon -g -h -p
                                            xxxxx console login:T sun -d /
root   260     1  0   Oct 13 ?        0:00 /usr/lib/snmp/snmpdx -y -c
                                          /etc/snmp/conf
root   270     1  0   Oct 13 ?        0:00 /usr/lib/dmi/snmpXdmid -s
                                            xxxxxx
root   282   260  0   Oct 13 ?        0:00 mibiisa -p 32788
root   269     1  0   Oct 13 ?        0:00 /usr/lib/dmi/dmispd
root 11379     1  0   Nov 06 ?        0:00 /usr/openwin/bin
                                          /fbconsole -d :0
root 11376   280  0   Nov 06 ?        0:02 /usr/openwin/bin
                                          /Xsun :0 -nobanner
                                          -auth /var/dt/A:0-8Iv0z_
root 11301   133  0   Nov 06 ?        0:00 /usr/dt/bin/rpc.ttdbserverd
root 11393  1377  0   Nov 06 ?        0:01 dtgreet -display :0
root 11377   280  0   Nov 06 ?        0:00 /usr/dt/bin/dtlogin -daemon
```

合わせると 60 数個のプロセスが動いています。私は、こういう場合、要らないものは止めるチューニング作業をします。

インターネットサーバの場合は、コンソールからの作業で十分ですからOpenWindows の daemon は止めます。かな漢字変換も必要ないので止めます。NFS は不要なので NFS 関係の daemon も止めます。RPC ですが、インターネットサーバでは必要ないので私はこれらを全部コメントアウトします。snmp はあっても良いのですが私の場合は使わないので止めてしまいます。プリンタも多分、インターネットサーバでは必要ありません。CD-ROM をマウントする vold も必要ありません。そういう変更でチューニングを行います。また、Mail サービスを行うサーバを構築するのでなければ、sendmail のオプションの「- bd」を抜いておくと sendmail 自身はメールを受けなくなるので、この方法で sendmail を止められます。

Solaris2.6 をそのような方法でチューニングしたあとの「ps –efa」の結果は次のとおりです。

Solaris 2.6 のチューニング後

ps -efa の結果

```
UID    PID   PPID C STIME TTY        TIME CMD
root     0      0  0   Sep 26 ?        0:00 sched
root     1      0  0   Sep 26 ?        0:18 /etc/init -
root     2      0  0   Sep 26 ?        0:00 pageout
root     3      0  1   Sep 26 ?      390:47 fsflush
root   146      1  0   Sep 26 ?        0:00 /usr/lib/sendmail -bd -q1h
root   214      1  0   Sep 26 ?        0:00 /usr/lib/saf/sac -t 300
root   104      1  0   Sep 26 ?        0:18 /usr/sbin/inetd -s
root    94      1  0   Sep 26 ?        1:10 /usr/sbin/in.named
root   109      1  0   Sep 26 ?        0:21 /usr/sbin/syslogd -n -z 12
```

```
root   126     1   0   Sep 26 ?         1:02 /usr/lib/inet/xntpd
root   136     1   0   Sep 26 ?        13:51 /usr/sbin/nscd
root   130     1   0   Sep 26 ?         0:16 /usr/sbin/cron
root   156     1   0   Sep 26 ?         0:01 /usr/lib/utmpd
root 19855     1   0   Oct 30 console   0:00 /usr/lib/saf/ttymon -g -h -p
                                              xxxx console login:-T sun -d /d
root   221   214   0   Sep 26 ?         0:00 /usr/lib/saf/ttymon
```

このチューニングによって、先に挙げた Solaris2.6 の標準インストールと比較してプロセスをここまで減らせるわけです。全体でも容量は 32MB 程度のメモリ容量で十分動かせます。

昔よく言われていたことですが、「swap サイズはメモリサイズの 2 倍確保する」というのがありました。けれども、これは過去の話です。

Solaris（2.6）の場合、スワップの許容容量は、ディスクに切った Swap 容量とメモリ容量を合わせた容量になります。私の場合、スワップ容量の設定をゼロにしてしまいます。

私が Solaris をインストールする場合、必要なものだけをインストールするという考えを基本にし、カーネルだけのコアインストールを選択して行っています。

ただし、最低限必要とするものだけのインストールではライブラリなども含めてパッケージが不足しますから、必要なものを、あとから pkgadd コマンドで追加します。合わせても 100MB 程度で済みます。たとえば、追加できるコンパイル環境のパッケージには以下のようなものがあります。

●system	SUNWsprot	Solaris Bundled tools
●system	SUNWtoo	Programming Tools
●system	SUNWbtool	CCS tools bundled with SunOS
●system	SUNWhea	SunOS Header Files
●system	SUNWarc	Archive Libraries
●system	SUNWntpr	NTP, (Root)
●system	SUNWntpu	NTP, (Usr)
●system	SUNWscpu	Source Compatibility, (Usr)
●system	SUNWlibC	SPARCompilers Bundled libC
●system	SUNWlibm	Sun WorkShop Bundled libm

Solaris をよく知っている人にはメモリ節約のこの方法もお勧めします。

ただし、サービスの止め過ぎには注意してください。止めてみたはよいが、OpenWindows は rpc 関係で影響を受けるケースがあります。私が実際に経験したことですが、nscd（Name Service Cache Daemon）を止めたら Apache は動いているのに Netscape Enterprise Server が起動しなくなりました。止め過ぎには注意しましょう。止め過ぎの原因が判らないときは、結局 1 個ずつ動かしてみて探すことになります。サービスの中には止めるとパフォーマンスを落とすものもあるようです。

3.2 FreeBSD の GENERIC カーネル

FreeBSD をインストールした直後のカーネルは、GENERIC なカーネル、要するにインストールがうまくいくような最大公約数的なカーネルになっています。このため、SCSI コントローラのドライバが 10 種類以上も組み込まれていたり、いろいろな種類のネットワークインタフェース用ドライバも入っていたり、CD-ROM ドライバ関連では最近使われていないものも入っていたりします。しかも、カーネル内部のテーブル類は非常に小さいため、サーバで使うときには望んだ機能を持たせるためにリコンフィグする必要があります。

カーネルのリコンフィグ作業では、不要なドライバは削除し必要なものだけを残します。私は、NFS、MSDOSFS、CD9660、ed0aic0、nca0 は抜いてしまいます。

あとは、テーブルを必要な大きさに増やす作業です。maxusers を増やすとユーザに関係した全部が大きくなります。そこで、一部だけを大きくしたい場合は、NMBCLUSTER や FD SETSIZE などを活用します。

あとは、LINT というファイルが「/sys/i386/conf」の下にありますので、LINT を眺めながら GENERIC を修正します。

FreeBSD のカーネルにおいて、当初の GENERIC カーネルと実際のリコンフィグ後のサイズを以下に示します。

```
$ ls -l kernel.GENERIC kernel
-rwxr-xr-x  1 root  wheel  1564800 Aug  5 03:57 kernel.GENERIC
-r-xr-xr-x  1 root  wheel   764284 Sep 29 08:45 kernel

$ size kernel.GENERIC kernel
text     data     bss      dec      hex
1294336  81920    91112    1467368  1663e8   kernel.GENERIC
589824   53248    51600    694672   a9990    kernel
```

もともとの kernel.GENERIC は 1.5MB ぐらいあります。これをリコンフィグすると、764KB ぐらいまで減ります。

3.4 Solaris の場合

Solaris はカーネルのソースが公開されていませんので、FreeBSD のように簡単にリコンフィグすることはできません。ただ、デバイス類は「boot -r」オプションを指定すると必要なデバイスのみロードされますから、デバイスドライバが多過ぎるような状態にはなりません。必要ならば、「/etc/system」のチューニングを行います。FD（ファイルディスクリプタ）を増やしたり、pty を増やしたり、共有メモリのサイズを変更したりします。このような情報は、Sun の配布ドキュメントの中には少ないのですが、AnswerBook の中に一部あるものと、Solaris-Ⅱ FAQ を参考にします。

Solaris のチューニングで、もしネットワークパラメータをチューニングしなければならない状態に陥った場合は、「/dev/tcp」というデバイスがあり、ndd コマンドを使ってパラメータを変えることができます。tcp パラメータのチューニングの際には、変数の種類は ndd コマンドで「usr/sbin/ndd/dev/tcp ?」と入力すると表示されます。ただ、変更が必要になるのは特別な場合です。

4 Server Software

4.1 WWW サーバ

WWW サーバは HTTP のリクエストを処理するサーバです。インターネットの中で最も稼働台数の多いサーバのひとつです。HTTP の処理では、TCP が接続されてクライアントからページのリクエストが送られてくると、サーバがそれに対してコンテンツイメージを返し、最後に TCP が切断されます。HTTP の処理はこれの繰り返しになるわけです。

実際のレスポンスまでの処理は、WWW サーバがリクエストの解析をして、リクエストがページの読み出しであるのか、CGI の実行なのか、ページが URL でマッピングされているのかなどを判断し、ページデータの読み出しや CGI を実行します。当然ページデータの読み出しではディスクアクセスが伴い、CGI のエンジンがプロセスを生成して、そのプロセスの実行結果を送り出すという処理を行っています。無視できないのがログの生成です。IP アドレスから FQDN(Full Qualify Domain Name)を知るために、ネームサーバ(DNS)への問い合わせも行います。この問い合わせは時間がかかるため、必要なときだけ行います。こういうことをしながらトータルのレスポンスを上げていきます。

4.1.1 WWW サーバボトルネックの要因

WWW サーバのボトルネックの要因となるのは、まずコンテンツデータの読み出しです。読み出しにはディスクの I/O が入ります。同じデータを読む場合はディスクのバッファにデータがあるので問題はありませんが、異なるデータを読み出す際にはディスクへのランダムアクセスが必要で、ディスクのシーク時間がボトルネックの要因になります。

あとは CGI プログラムの実行です。これが実は、くせ者です。CGI には fork という作業が必要です。CGI プログラムが大きいと、CPU の性能不足によって実行に時間がかかったりします。メモリが不足してスワップが起き、実行が遅くなることもあります。

また、CGI が書かれている言語が C か Perl かで CGI の実行時間が異なってきます。まれに Perl で書かれているほうが速いこともありますが、Perl は初期化に時間がかかるため、一般に C で書かれたほうが速い処理ができます。

このほか、ログ書き出しの負荷がボトルネックの要因になります。たとえば 1 日に 100 万ヒットする WWW サーバでは、ログを 1 秒間に 10 行以上書かなければなりません。このため、ディスクが遅いと記録できないこともあります。DNS を調べて FQDN で記録する場合も、ログの記録が遅くなる原因になります。

また、大きなコンテンツの転送時間も問題になります。転送の際のネットワーク環境やサーバの転送容量などがネックとなってコンテンツの転送に時間がかかる場合には、ISP のハウジングサービスを借りることによって解決されることもあります。

4.1.2 WWW サーバの 1 リクエストの負荷

WWW サーバでリクエストの処理を考える場合、必ずサーバへの負荷を考えなくてはなりません。

WWW サーバの 1 リクエスト当たりの負荷は、「リクエスト数×処理時間」で、この両者の積をよく考えなければなりません。

たとえば 1 リクエストの処理を、終了までに 1 秒かかり、その処理にメモリを 1MB 使用し、平均で 1 秒間に 10 リクエストがある場合、リクエストを処理しきるためには 10MB の実装メモリが必要になります。毎秒 100 リクエストがあるとすると、メモリは 100MB 必要です。同じく 1 秒間に 100 リクエストがある場合でも、CPU が 10 倍速ければ、1 リクエスト当たりの処理時間は 0.1 秒と 10 分の 1 になり、理論上では、メモリは 10MB で済むことになります。

4.1.3 ネットワークの転送速度

やはり無視できないのはネットワークの転送スピードです。WWW サーバのネットワーク転送速度を速くしても、クライアント側のネットワーク転送速度が遅いままでは、1 クライアント当たりの転送時間は変わりません。

これらの諸条件によって、WWW サーバが同時処理するプロセスの負荷が変わってくるわけです。

4.1.4 主要な WWW Server Software

WWW サーバでメジャーなソフトウェアと特徴を次に示します。

●CERN	リクエストごとに fork
●NCSAhttpd	高機能で CERN より高速
●Apache	fork 済みのプロセスに fd passing NCSA httpd も採用
●phttpd	fork の代わりにスレッドを採用
●thttpd	select を使ったループで処理

4.1.5 CERN Server

CERN は、WWW コンソーシアムが結成される以前に WWW サーバのサンプルインプリメンテーションとして作られました。Proxy 処理と fork 処理、キャッシュに特徴があり、Proxy サーバと WWW サーバが 1 つの設定で済むことなどが便利で、一時期広く使われていました。ただ、ソースコードのボリュームが大きく、1 リクエストごとに fork 処理するため、レスポンスが増えると時間がかかり過ぎることから現在はほとんど使われていません。

4.1.6 NCSA httpd

NCSA httpd は、NCSA(National Center for Supercomputing Applications)で作られたサーバソフトウェアで、やはり fork 処理が特徴です。CERN よりも機能が高かったために一時期広く使われました。URL のリダイレクト、アクセス制限の設定が柔軟な点も優れています。実際に計測しましたが、CERN のサーバよりも 60%高速でした。

4.1.7 Apache WEB サーバ

Apache は、現在世界で一番稼働台数の多い WEB サーバです。開発者は正確な動作を重視したインプリメントを行ったと述べており、スピードよりも正確に動作することを優先しています。実際に使用してみましたが、十分に高速です。

Apache も fork 処理を行いますが、あらかじめ fork したプロセスを複数個用意しておいて、それぞれのリクエストに応じて子プロセスを順に起動していくため、CERN や NCSA httpd よりも高速です。また、豊富なモジュールがあり、バーチャルホストにも対応しています。

4.1.8 phttpd

phttpd は、スレッドを利用してリクエストを処理するのが特徴です。スレッドは fork に比べるとはるかに軽いため、非常に高速な処理が可能です。Netscape Enterprise Server の Solaris バージョンも内部でスレッド処理をしています。ただ、スレッドがきちんと動く OS が少ないために移植性に多少難があり、現状は Solaris 以外で動いている例はありません。

4.1.9 thttpd

面白いのが thttpd で、私はこれが好きでよく使っています。複数のコネクションを select で同時処理するため、一切 fork 処理をせず非常に高速です。しかし、機能的には基本機能だけで、URL のリダイレクトはできません。ちなみに、世界で 6 番目に多く使用されている WWW サーバソフトウェアです。

4.1.10 CGIについて

ここで、WWW サーバが関与する CGI について説明します。

CGI は、Common Gateway Interface のことで、WWW サーバから外部プログラムを実行し、動的にページイメージを作成して、その結果を表示するものです。

その身近な例が WebBSS や WebChatt です。WebBSS では、Perl などと組み合わせて CGI が広く使用されています。

CGI の処理の流れでは、外部プロセスを生成しなければならないため、WWW サーバが fork 処理をして、その後に exec を実行します。

fork 処理をすると 1 つのプロセスが 2 つになり、データ領域のコピーが発生することから、UNIX での重い処理(負荷がかかる処理)と言われています。ただし copy on write があればそれほど重い処理にはなりません。

exec は実際のプログラムの実行となります。fork から exec に移るまでに、少なくても数ステップの命令を実行するため多少なりとも時間がかかります。この時間が問題になります。

copy on write がなければデータサイズが大きいほど処理は遅くなりますが、copy on write があっても問題が発生することがあります。

たとえば、Netscape Enterprise Server の場合ですが、このサーバはスレッドベースのサーバと言われています。しかし、HTML の処理はスレッドですが、CGI の処理は fork です。しかもコード量が大きくて、何もしなくても 10MB 程度のプロセスサイズがあり、1 リクエストが増えるごとにそのプロセスのために必要なメモリが 800KB 程度増えます。リクエストが 100 を超えると 50MB を超えるメモリが必要になります。fork から exec までの時間にリクエストが集中してきた場合、そうした 50MB を超えるプロセスが fork したら、瞬時にメモリを消費します。実際にはそれに必要なメモリを実装したシステムを実現するのは無理です。Apache は方式が独自のためこうした問題は回避していますが、その他の fork 処理ベースの WWW サーバでは同様の問題があります。

4.2 NEWS サーバ

NEWS サーバはインターネットのサーバの中では最も負荷のかかるサーバで 1 日に 50GB 以上のトラフィックを処理しています。NEWS サーバは他のサーバとの記事交換と、エンドユーザ向けの購読、投稿サービス、記事のコントロール処理を行います。

4.2.1 記事の受信

NEWS サーバが記事を受信する場合、他のサーバからの記事の受信時には記事の重複の検索を行い、スプールに書き込みます。別のサーバへ転送が必要な場合には、送信ファイルを生成します。記事の重複の検索や、スプールへの書き込みにはディスク I/O を頻発します。最近のトラフィックは膨大で、6Mbps の回線がないと対応できません。T1 では１日 15GB がやっとで fullfeed に対応できない状況にあります。

4.2.3 記事の送信

記事の送信は、他のサーバへの記事の送信および記事を送信ファイルから読み込んで送信するという処理を行います。スプールから読み込む際に、やはりディスク I/O の量が増えます。

4.2.4 ユーザの購読用サーバ

ユーザ購読用サーバは、管理する情報が増えます。ニュースグループと記事番号の管理、overview 情報、history 情報からスプールの記事への対応、cancel 処理の問題もあります。

4.2.5 従来の INN(before1.x)

NEWS サーバソフトウェアに INN があります。従来の INN はニュースプール形式に ufs(UNIX File System)を使用していました。記事番号をそのままファイル名にしていて、サーバはこのファイル名で送る形を取り、ニュースグループをそのままディレクトリにしていました。ですから、news.software.nntp の 350 番目の記事は /var/spool/news/news/software/nntp/350 となります。

この形式では、ファイル数が増えた場合、1 つのディレクトリに置かれるファイル数が多くなるため、記事が増えるほど検索時や送信時にそのファイルを特定するまでの時間もかかることになります。

また、cancel(記事の取り消し)処理のとき、取り消し対象となった記事が/var/spool/news/control のディレクトリに置かれている関係で、/var/spool/news/control まで行ってファイル削除を行うので、多くの階層を経ていかなくてはならず、コントロールの記述そのものも次第にボリュームが増えてしまいます。膨大な最近のニュース量では、ufs がボトルネックになります。

4.2.6 ufs のボトルネック

ufs では、ファイルを検索するときに 1 ファイルずつ順次すべてのファイルを調べるようになっています。ファイルを削除する場合よりもファイルを作る場合のほうが大変で、ディレクトリ内に同じファイル名が存

在しているかどうかをリニアサーチしてみなければなりせん。ファイルが 1000 個あれば、サーチするのに少なくとも平均でその半数をサーチする必要があり、かなりの時間がかかります。これがボトルネックとなります。

UNIX はほとんどが ufs ですが、市販品では vxfs もあり、vxfs はハッシュ構造を持っているのでこうした問題は生じません。また Windows NT もハッシュ構造になっています。

4.2.7 Diablo

最近登場した NEWS サーバのソフトウェアに、Diablo があります。もともと配送専用のサーバで、ISP のバックボーンとして、隣から受け取った記事をさらに隣へ送るという機能を高速に行うよう作成されています。高速である理由は、キャンセル処理をしないために余計なディスク I/O が発生しませんし、active や newsgroups などのファイル(必需品ではあるのですが)が無いからです。

しかし､最近 Diablo にも購読用サポートが入りましたので､現在は active があります。当時の話ですが、INN 1.x の 5 倍から 10 倍のパフォーマンスがありました。これは、ニューススプールを独自の形式にしたためです。INN のバージョン 1 の場合 1 つの記事が 1 個のファイルになっていてディレクトリに大量の記事が置かれるのですが、Diablo の今のバージョンでは時間と関係したディレクトリの中にニュースの記事の巨大なファイル(100 個から 1000 個)を作るため、パフォーマンスが大きく向上しています。

4.2.8 現在の INN(version2.x)

INN もバージョン 2.0 から CNFS(Cyclic News File System)を導入しました。あらかじめ巨大なファイルを作っておき、頭から順に使っていき、最後まで行ったらまた前から使うという一種のデータベースエンジンの機能を持っています。これによって cancel 処理が劇的に改善されていますので、Diablo と変わらないくらいのパフォーマンスになっています。

4.3 Mail サーバ

メールサーバには 2 種類のサーバがあります。ひとつはメーリングリスト用のメールサーバで、速く、大量に、より多くのユーザにメールを配信するということが目的です。

もうひとつは、POP3、SMTP サーバと呼ばれる PC ユーザのダイヤルアップ用のサーバで、1 台のマシンでより多くのユーザをサポートするのが目的です。

4.3.1 メーリングリストサーバ

メーリングリストサーバとしての sendmail は配送が非常に遅かったの

で、ある記事の配送が終わる前に別の人が返事を書く可能性がありました。sendmail における従来の問題点です。最近、それを改善するために qmail や sendmail + WIDE patch + smtpfeed が比較的多く使われています。

4.3.2 Sendmail の特徴

従来、メールサーバというと sendmail しかありませんでした。sendmail は基本的に逐次配送しか行いません。sendmail + WIDE patch というのが昔のバージョンです。WIDE patch を当てると少し速くなるのですが、それでも配送部分については順にしか配りません。

逐次配送ですと、メールが送り先に届かないケースが発生した場合、それを判断するまでに少し時間がかかります。また、ネットワークの状況が悪くてパケットロスが起きる場合には TCP で再送が行われ、これも時間がかかる要因になります。1000 人位の配送先があった場合、途中に少し遅れたメールが数個入っているだけで、最後の人のメールに対してはものすごく遅くなります。最初のメールが投げられてから最後のものが送り終わるまでに、3 時間もかかったりします。sendmail の良い点は、同一 MX 先の相乗りができることです。

user1@foo.co.jp や user2@sh.foo.co.jp のように、foo.co.jp と sh.foo.co.jp が同じ MX 先のホストであった場合は、1 通の配送で終わります。もし 100 アドレスあったとしても、MX の先が 1 つだとすると、投げるのは 1 回で済むので速いのです。

4.3.3 qmail の特徴

最近話題になっている qmail は、もともとは高速配送という目的ではなく、sendmail のセキュリティホールがあまりにひどくて、インプリメントもこれではいけないと思った人が作ったプログラムです。複数の小さなプログラムを組み合わせて用途別に処理します。

qmail-smtpd、qmail-inject、qmail-remote などの細かいプログラムが数多くあります。かつて、sendmail はセキュリティホールが数多くあり、たびたびバージョンアップしなければならなかったもののひとつだったのですが、最近はあまりセキュリティホールは出ていません。

qmail の面白いのところはセキュリティホールの発見に 1000 ドルの賞金がかかっていることです。処理的には sendmail よりはるかに軽くて高速です。

ただ、同一ドメインであっても 1 通ずつ配送するという問題があり、usr@foo.co.jp、usr1@foo.co.jp、usr2@foo.co.jp……と 100 種類のアドレスがあった場合には 100 通のメールを投げてしまいます。制限はできるのですが、それぞれの宛先に対して 1 個のプロセスが起動しますので、同時に 100 個のプロセスが動く事態が発生します。ディスク I/O が絡んできますから、ハードディスクの構成をよく考えないと、ディスクの読み出しでボトルネックになることがあります。

4.3.4 sendmail + WIDE patch + smtpfeed

sendmail(たとえば 8.9)+ WIDE patch + smtpfeed という組み合わせが最近の高速なメール配送ではよく使われている手段です。smtpfeed は、sendmail が 1 通ずつ逐次配送するのを改善するために外部 mailer として sendmail から呼び出され、実際の配送を分担します。ですから、アドレスなどの解析は行わずに、sendmail からもらったアドレスに対して投げるだけで、sendmail と LMTP(Local Mail Transfer Protocol)というプロトコルで通信します。

この組み合わせの良いところは、MX の相乗りを行うことです。つまり、select を使って複数に同時に配送を行うのです。A のアドレスにメールを投げている間に、B のアドレスに対してもメールを配送します。そのおかげで、極めて高速な動作が可能です。実際に DX4 でやってみたのですが、1600 アドレスを与えたところ 3 分以内に 90%のメールを配送しました。非常に高速です。もちろんこの数字を出すためには、ネットワークのロケーションもそれなりに良くする必要があります。これは、1 個のプロセスで全部処理します。先ほどの qmail の場合はアドレス数だけのプロセスが立ち上がりますので、気をつけないとプロセス制御やメモリの問題が出てきます。

smtpfeed はすべてをメモリ中で処理しますので、宛先が増えるとメモリの使用量も増えます。

4.3.5 POP3、SMTP サーバ

POP3 と SMTP サーバはエンドユーザ用のサーバです。SMTP は普通のメールサーバと変わりません。

POP サーバには、一般に qpopper というプログラムがよく使われます。

/spool/mail ディレクトリにユーザ数分のファイルが置かれ、アカウント数が多いと同一ディレクトリに大量のファイルが作られるため、ファイルを探す際に先ほど説明した ufs でのリニアサーチが発生します。

4.3.6 qpopper の注意点

悪いことに、qpopper はデフォルトのままだとリクエストがあるとメールボックスを 1 行ずつ同じディレクトリのテンポラリファイルにコピーし作成します。ユーザ数の倍のディレクトリエントリを消費するので大変です。さらに、接続終了時に元のファイルに書き戻すので、ufs によるディレクトリのリニアサーチの問題も効いてきます。ただし、qpopper はソースが公開されているので、テンポラリファイルを別のところに置くなどの改造をするとパフォーマンスを上げることができます。

4.3.7 運用開始後の監視

次は運用開始後の監視です。インターネットサーバの場合は、必要とするレスポンスが十分に出ていれば、ぎりぎりでも良いというのが基本にあります。さまざまな CPU の状態を監視するプログラムがありますの

で、それらを利用して定期的にチェックすることが重要です。

次の図 3 は、実際に監視プログラムを使用して私が NEWS サーバの実機で負荷と応答時間の関係を調べたものです。

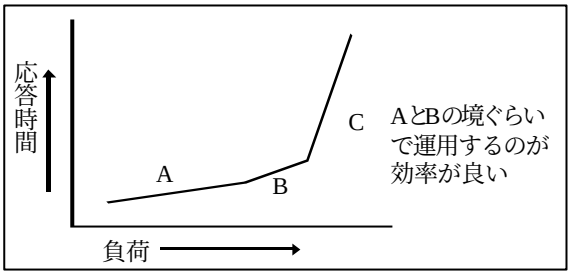

図 3：負荷と応答時間

一般に、負荷が増えれば応答時間は遅くなります。グラフも右上がりの傾向を示しています。興味深いのは、A の部分に比べて B の部分の右上がりの角度が大きくなり、さらに C では急角度で上がっていることです。

A はシステムリソースに十分余裕がある段階、B はやや余裕がなくなりはじめた段階、C はリソースの全体または一部に過負荷がかかった段階と考えてよいでしょう。

ここで注目するのは、B から C へ移ると急に応答時間が遅くなることです。ちなみに、C の段階で CPU の処理能力にはまだ余力は十分ありましたが、実際には B の段階に比べて処理量がほとんど増えていなかったことが監視プログラムでわかりました。この原因は、むしろ、ディスクへのランダムアクセスが増えて、CPU が処理したくても待たされるという状態に置かれたためでした。

また、NEWS サーバへのリクエスト数の差は、B と C とでは極端に差があったわけではなく、むしろちょっとした差で応答時間が急激に悪化することが明らかになったものです。サーバを運用する際には、実際に監視を行って、A と B の境界あたりで運用すると効率が良くなると言えるでしょう。

5　さらなる高負荷への対応－ラウンドロビン

さらに高負荷になって、インターネットサーバが 1 台で対応できなくなったらどうするかについて、その手法となるラウンドロビンについてここで説明します。

5.1 ラウンドロビン DNS

ラウンドロビン DNS は複数のサーバを 1 台に見せる手法のことで、最近では WEB サーバで多用されています。ただ、サーバによってはラウンドロビン DNS ができないものもあり、データベースサーバでは、データベースを同期させなくてはならないため簡単にはできません。また、NEWS サーバも難しいです。ヤフーの場合、日本のサイトだけでも 10 台のサーバがあります。goo、Infoseek などの検索エンジンを持つサーバや、asahi.com などアクセスが非常に多い新聞社などの WEB サーバの多くで使われています。

ラウンドロビン DNS には､A レコードによるラウンドロビンと CNAME によるラウンドロビンがあります。

一般的に多いのは、A レコードによるラウンドロビンです。3 台のサーバがあるときに A のゾーンに 3 つとも設定します。ゾーンファイルに設定するときに、以下のように設定します。

●www	IN	A	10.10.10.1
	IN	A	10.10.10.2
	IN	A	10.10.10.3

つまり、最初の問い合わせには 1 番目、次には 2 番目、その次には 3 番目を先頭にするわけです。

●A の問い合わせには順番を入れ替えて返す
- 1 つめの問い合わせ　10.10.10.1,10.10.10.2,10.10.10.3
- 2 つめの問い合わせ　10.10.10.2,10.10.10.3,10.10.10.1

ネームサーバへの問い合わせがあった場合、3 つとも返しますが、その際に 3 つの順位を順次入れ替えて答えます。

●クライアントは最初の A レコードから順に試す

ユーザは、最初にアクセスできたところに次回からもアクセスする習性があるため、3 台への負荷は分散されることになります。

一方、CNAME によるラウンドロビンは、CNAME は別名を示す設定で実は禁止されていた設定を流用したものです。ネームサーバの bind 4 などでは実際に設定ができたので、それをラウンドロビンに活用したわけです。ネームサーバの新しい bind 8 では、デフォルトでは禁止されていますので、options のフィールドに multipl-cnames yes; という記述を加えることが必要になります。

ゾーンファイルの設定例は以下のとおりです。

●www	IN	CNAME	www1
	IN	CNAME	www2
	IN	CNAME	www3
●www1	IN	A	10.10.10.1
●www2	IN	A	10.10.10.2

●www3　　IN　　A　　10.10.10.2
●bind 8 では multiple-cnames　yes; とする

CNAME ラウンドロビンでは、クライアントがキャッシュネームサーバに問い合わせると、キャッシュネームサーバはマスターネームサーバへ問い合わせます。マスターネームサーバは CNAME の 1 つだけを返し、キャッシュネームサーバはその結果をキャッシュします。このため特定のクライアントには特定のサーバにだけアクセスすることになります。きれいな分散にはなりませんが、分散効果は得られます。

A コードによるラウンドロビンと CNAME によるラウンドロビンの比較を次の表 2 に示しておきました。

表 2：ラウンドロビンの比較

	A	CNAME
マスターからキャッシュへ	全てのA	1つのCNAME
キャッシュするもの	全てのA	CNAMEとA
クライアントへ返す	毎回違う順のA	同じA
クライアントのアクセスするサーバー	全てのサーバーを順にアクセス	特定のサーバーをアクセス

6　実例

6.1　news.nspixp.wide.ad.jp[1996/10]

WIDE Project の商用 IX(Internet eXchange)の NEWS サーバで、私が手掛けました Nspixp（news.nspixp.wide.ad.jp）を例に説明します。

ニュースの交換というのは、基本的には 1 対 1 で交換しなければいけません。たとえば 3 つのプロバイダ、A、B、C を考えた場合に、それぞれニュース交換しようとすると、A と B、B と C、C と A がそれぞれ nmtp の設定を行う必要があります。ただ、IX のような場所ですと、数が増えるとメッシュ状に増えてしまって大変なので、間にマシンを 1 台置いて、そこと交換すれば良いのではないかと考え、構築してみました。

1996 年の秋に作成したサーバの構成は次のとおりです。

●CPU　　133MHz 版 Pentium
●メモリ　　128MB

●HD　　2GB を 2 台と 4GB を 1 台

2GB のディスクは 1 台をシステムに、1 台をニュースの history に、4G のディスクをスプールに置きました。SCSI は、同時に書き込めるというパフォーマンスを期待して、2GB の 2 台のほうに 1 チャンネル、スプール専用に 1 個を使ってみました。最初はさすがに良いパフォーマンスで動いていたのですが、時間が経つにつれていろいろと問題が起きました。ソフトウェアは Diablo を使いました。しばらく運用していると、ニュースが送れなくなったのです。何が悪いのだろうと、メモリ、ディスクを調べていたのですが、最終的に CPU がネックではないかと考え、vmstat で調べたら処理能力は 50%余っていました。どうも変だと思い、CPU を 200MHz 版 Pentium Pro に交換したところ、速くなりました。

その他、netstat の結果で、incoming が多いと outgoing が減っているし、outgoing が多いときは incoming が減っています。合計すると Ethernet の帯域いっぱいではないかということになりました。Ethernet には 100Mbps 対応のインタフェースを使っていたのですが、調べたところ、Ethernet スイッチの設定が間違っていました。100Mbps での設定のつもりでしたが、実際には 10Mbps で通信していました。正しく設定したところ、1 回のパフォーマンスが上がりました。つまり、Ethernet がボトルネックになっていたわけです。

FreeBSD でのチューニングオプションとして重要な mount option に noatime があります。それを試してみました。

UNIX はファイルを open して close するだけでも i ノード上でアクセスタイムの更新を行います。

この数が 1 個や 2 個ならよいのですが、NEWS サーバのようにスプールを大量にアクセスすると i ノードの更新時間が無視できないため、noatime を追加してみたわけです。その結果は、観測した限り少しだけパフォーマンスが上がりました。もっと効果があるとされるオプションに async があります。できるだけディスクに書き込みにいかないというオプションです。

async の状態でシステムがクラッシュするとファイルシステムに膨大なダメージが起きると考えられますから、普通は勧められません。ただ、ニューススプールだったら大丈夫と考えて試してみました。しかし、これは全然効果がありませんでした。まだメモリが 128MB のときです。async で書き込みを遅らせたところで、結局バッファリングに使えるメモリがあまり余っていなかったので効果がなかったわけです。

この後、Diablo で shm(シェアードメモリ)を使うように修正してみました。shm はニュースの SPAM(あちこちの記事に大量にポストする)のチェックに使っていたと思います。

config のミスで shm がうまく使われていなかったのを使えるようにしたのですが、かなり効果がありました。Diablo は SPAM の情報をハードディスクに書くため、shm を使うと大きな差が出ます。incoming は 12GB 位だと 1 日当たり 60 数 GB ぐらいの outgoing を出すだけの能力はあります。ただ、incoming が 18.57GB/日になってしまうと、outgoing が減ってしまうのです。シークが大きく増えてスプールのボトルネック

が発生するのです。2 時間に 1 回 expire をかけて記事の古いものを捨てるのですが、それでもものすごい量で、こちらへ来た記事をよそへ配る前に expire によって無くなってしまう事態が発生し、outgoing が 43.17GB に減ってしまいました。その後、メモリを増やしてみたら、expire で当初 30 分位かかっていたのが、バッファリングが効くようになり半分に短縮できました。ディスクバッファを大量に必要とするようなプログラムでは、メモリをたくさん積むのが有効です。

6.2 news.nspixp2.wide.ad.jp[1998/1]

先ほどのマシン news.nspixp.wide.ad.jp があまりにスプールが小さくて、もっとスプールの I/O を稼がなければならないということで作ったものが news.nspixp2.wide.ad.jp です。98 年 1 月に構築しました。構成は次のとおりです。

- ●CPU　333MHz 版 PentiumⅡ
- ●メモリ　384MB
- ●HD　9GB を 2 台

CPU は、発熱量が少ないのと入手し易さを考えて選択しました。発熱というのは結局マシンの寿命を縮めるもとですので、発熱の少ないものに越したことはありません。この CPU は、当時はまだ出たばかりだったのですが、今後を考えると入手しやすくなるだろうと判断しました。

というのは、CPU が壊れたときは交換しなければいけないので、運用を考えると入手しにくい CPU を使うというのは良くないのです。メモリは、先ほどの NEWS サーバが 256MB まで載せたのですが、それより多くのメモリが必要だということで、この時のマザーボードの限界（128MB の DIMM を 3 枚）まで載せて 384MB にしました。スプールの I/O を稼ぐために当時の 9GB のディスクを、ccd を使ってストライプにしました。接続先ポートの関係で FDDI インタフェースを使っています。

6.3 news.nspixp2.wide.ad.jp の実力

news.nspixp2.wide.ad.jp は、31 の ISP と接続され、1 日当たりのトラフィックは、incoming の記事数が 55 万通から 80 万通、18GB から 23GB というとてつもなく多い量です。この数字は 1 年前のものですが、このレベルが出てくるためには、回線は 2Mbps ぐらいないとだめです。

つまり、T1 では送り切れないと言われている量です。1 日に 330GB の outgoing の能力を持っています。98 年 12 月 6 日に同年の最大トラフィックが発生しましたが、1 日に incoming が 25.7GB、outgoing が 361GB になりました。普段は FDDI の 40%位の帯域をコンスタントに使っていましたが、ピーク時は FDDI の帯域を使い切っています。

必要であればこういう大規模なマシンを作れば良いのですが、このマシンは最近バテ気味で、時々破綻をしています。

6.4 sh.janog.gr.jp[1998/8]

次に、JANOG のメールサーバとして 98 年 8 月に構築しました sh.janog.gr.jp の事例を紹介します。JANOG のメールサーバと WWW サーバです。

●CPU	100MHz 版 DX4 ODP
●メモリ	16MB
●HD	SCSI 2G1 台
●MB	ISA/VL
●OS	FreeBSD RELEASE 2.2.7
●www	thttpd2.04
●mail	sendmail 8.9.1a + WIDE patch 3.1W + smtpfeed0.90

このシステムは、JANOG 内で相談して NTTPC Communications Inc.のハウジングサービスにマシンを置くことになり、お金をかけずにという設計目標のもとに作りました。

ですから､ごみになる直前の DEC PC を拾ってきました｡CPU は 66MHz 版 486DX だったので、少しでも速くなればと思い DX4 ODP をどこかから拾ってきました。メモリは最初に拾ってきた状態で 16MB だったのですが、OS が 1MB ぐらい、動かすプログラムを考えても足りるだろうということで 16MB のままです。ハードディスクは SCSI 1 台で 2GB を入れてあります。SCSI のコントローラは Adaptec の 1542CF を使っています。マザーボードは ISA/VL です。

JANOG のメールを受けている方の場合、最初の人がポストしてから手元に届くまでが結構速いと思うのですが、現状 5 分ぐらいでほとんどのメールを配り終わっています。

メモリ 16MB で現状十分なのは、smtpfeed はピーク時には 2MB ぐらいメモリを使いますが、普段は sendmail しか動いていなくて､sendmail が待っているだけだからです。thttpd も 1MB しか使いませんし、コネクションが増えても使用メモリがそれほど増えません。また、fork も行わないので、ほとんどメモリを圧迫しません。

何が一番圧迫するかと言いますと、チェックに入ってログインする際にウィンドウをいくつか開くと、そこで起動する bash や tcsh の方がメモリを圧迫します。普段はサーバですからログインする必要もないので、その分のメモリは空いています。現状は 16MB で十分です。

それでも、現在 1660 人ぐらいのメールアドレスに配っています。まだ WWW サーバとしてはあまり活躍していませんが、良いパフォーマンスを示しています。

6.5 Solaris 2.x での例

今度は最近構築したサーバの例です。もともと Ultra 1(メモリ 512MB)のマシンがありました。ここで 1GB メモリを欲しいという要求があったのですが、1GB のメモリは高価です。

何でそんなにメモリが要るかと言いますと、このサーバはメモリ 512MB で、60MB や 70MB の数個のプロセスがぎりぎり動いているのですが、もう少し何とかしたいのでメモリを増やしたいというのです。

CPU が Ultra 1(UltraSPARC の 167MHz)ですが、この CPU にメモリを 1GB も積んでも大丈夫かと考えたのです。大丈夫かというのは動く動かないという問題ではなくて、投資に見合うかというところを検討しないといけないわけです。

CPU の能力に対してあまりに巨大なメモリを載せても、バランスが崩れていると感じるわけです。考えなければならないのはこのことです。

6.6 Solaris 2.x での例 SPARCengine Ultra AXi の導入

これでは良くないと思い、いろいろ調べた結果、Sun が OEM 用に出しているマザーボードを見つけました。これは Sun の OEM 用プロダクトです。OEM ですから紙に印刷されたマニュアルはなく、PDF のファイルがあるだけです。そのマザーボードは、ATX のボードサイズに 300MHz の UltraSPARC-IIi を実装していました。現在の CPU に比べて一回り高速になります。

これ位の CPU であればメモリを 1GB 位積んでもバランスがとれると私は思いました。

ボードだけなので、自分でケースも買わなければいけませんし、もちろんメモリもきちんと選ばなければなりません。

オンボードにネットワークインタフェースと SCSI インタフェースが付いています(今売っている Ultra5 には SCSI インタフェースはありません)。ですから、Ultra10 よりも良いのではないかと思います。このシステムはサーバですから画面はどうでも良いので、秋葉原で売っている 7000 円位の PCI のビデオカードで十分です。モニタも買う必要はありませんでしたし、CD-ROM ドライブはインストールのときしか使わないので買いませんでした。

ラックマウントのケースにしておくと後々便利だと思ってラックマウントのケースを買い、大体 100 万円位で作成できました。

もちろん自分で組み立てなければいけないため、PC を組み立てられない人や Sun に詳しくない人にはお勧めできませんが、Sun を長く使っている人だったらこういう方法もあると思います。こういう導入の仕方というのも考えられるということです。もちろん保守の問題などはありますが、価格面で非常にメリットがありますからサーバ向けには良い選択かもしれません。

6.7 まとめ-高速化のための TIPS

とにかくシステムのボトルネックを探す、あるいはシステムを作るときにはボトルネックは作らないように何とかすることです。

重要なのは、あるアプリケーションを動かすと、そのアプリケーションはどういう挙動をして何をするか、これを理解していないと、思わぬところにボトルネックができてしまいます。理解していたはずでも、ちょっと失敗するとボトルネックになりますから、できる限りボトルネックは作らないようにします。

8-2 の法則という言葉があって、10 割の仕事があったとしても、そのうちの 2 割が本当にボトルネックになる、時間のかかる部分です。その 2 割の部分を改善すれば、全体としてはスピードアップを図ることができるということです。実際にそのとおりです。

面白いのは、インターネットサーバでは、CPU よりはディスクアクセスにボトルネックが起きます。

ですから、先ほどの JANOG サーバが良い例で、CPU は DX4 で極めて非力で、インターネットによる接続も普通の Ethernet のネットワークカードで ED0 というドライバを使って 10Mbps でつながっています。

エンドユーザの使用している回線は普通 64Kbps や 128Kbps ですから 10Mbps のレベルは確かにすごいわけです。sh.janog の場合ですと、メールを配る瞬間はその 10Mbps の Ethernet をフルに使って、大体数秒間それが続きます。その間 CPU のアイドルは全くなくなりますが、メールが 3 分程度で配れれば、その間どんなに CPU が忙しかろうがそれで良いのです。

smtpfeed の場合はディスクアクセスがあまり起きないのでボトルネックにはなりません。

ボトルネックというテーマはなかなか難しいテーマです。

カタログ性能だけでなく、実際に使ってみて速いディスクを選ぶことです。私の経験からすると、カタログスペックと実際のスピードは別のようです。カタログスペックではすごく速そうだけれども、使ってみるとどうも速くないというのがあります。

ハードディスクのアクセスを減らすというのが高速化のためには良いことです。1 つはメモリを十分に積んでバッファを効かせることです。極端な場合は async option を使ってみる方法も必要だと思います。また、1 台当たりのシークを減らすという方法で複数台のハードディスクを利用する方法もあります。

たとえば、WEB のコンテンツとログは同じディスクには置かないという話です。同じディスクに置くよりは、違うディスクにしておいたほうがヘッドの動きが減ります。容量ぎりぎりまで使わないということもあります。ufs の特性ですが、空き容量が 2 割のディスクと空き容量が 3 割のディスクを比較すると、私の計測した限りではスピードが違います。ですから、ディスクは余裕を持って使う、もっと極端な使い方になりますが、パーティションを切って使う方法もあります。

たとえば、8GB のディスクを持ってきても 2GB くらいしか使わなければ、シークの量は 4 分の 1 程度に減るのではないかという計算ができます。そこで、2GB でパーティションを切ってしまって使うという方法で

す。そうすると、ヘッドのシーク量は物理的に制限できますのでかなり速くなります。その他、ハードディスクのスピードを上げるためには転送レートを上げます。

コントローラは複数台用意して CPU の負荷を減らすという方法もあります。

もうひとつあります。私はこれが好きなのですが人間を鍛えることです。

ハードウェアだけ頑張っても、こういうノウハウは経験値が重要です。私はふだん遅いマシンでテストします。速いマシンだと見えないものが遅いマシンだと気がつくことがあるのです。たとえば速いマシンだったら 0.1 秒以下の差になってしまって全然見えないところが、遅いマシンを使ってみると 10 秒の差になったりします。本当に必要なときは速いマシンを買えば良いのですが、普段は遅いマシンでいろいろテストしてみるというのがひとつの例です。

他には、実際には難しいのですが、条件の違うマシンをいろいろ用意して同じことをやってみます。同じことというのは、たとえば、ちょっと大き目のプログラムをコンパイルして時間を計ったり、色々なベンチマークを走らせてみたりするのです。結局、結果に違いが出るわけですから、その違いがどこから出てくるかを見極めます。それを今度は速いマシンでやれば、より速くなります。

ノウハウというのは聞いただけではなかなか身につかないものですが、実際に痛い目に遭うとよく身につくと思っています。

次はシステムを構築するときのポイントですが、目的をよく見極めないといけないということです。何をするためのサーバなのかを見極めることです。WWW サーバなのか、NEWS サーバなのか、DNS なのか。スピードも重要ですが、コストも重要で、無制限にお金をかけるわけにはいきません。電力の問題もあります。あまり電気を食わないほうが良いわけです。

それと場所です。これも重要なパラメータで、学校のような場所だと比較的余裕はあると思うのですが、たとえば、プロバイダのハウジングサービスを借りるときですと、やはり小さく作っておかなければ同じ場所にたくさん置けません。私はよく「置けなくなったら負けである」という言葉を使っています。要するに、あるスペースがあったときにそこに 1 台しか置けなくても良いのか、10 台置ける方が良いのかというので違ってきます。同じ能力だったら体積は小さい方が良いですし、電力も少ないほうが良い訳です。あとは予算です。これはやはり無視できませんが、予算について言う人は少ない気がします。

結局チューニングというのは特別なことではなく、当たり前のことを当たり前に全部やれば確実に速くなるのです。ですから、当たり前のことの積み重ねですので、どこが悪いのかというのをひとつひとつ細かくチェックして、それを改善する努力をします。ひとつひとつで 5%ずつ改善すれば全体で 10%改善できるということもあります。積み重ねる努力が大切です。