修 士 論 文

システム構成要素間の相互作用を考慮したクライアント・サーバーシステムの性能解析

指導教官　日比野 靖 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

福田 次良

1998 年 2 月 13 日

要 旨

クライアント・サーバーシステムの一つとして、分散ファイルシステムをとりあげ、各々の部分的な変動がネットワークシステム全体の系にどのような影響を及ぼすかをシミュレーションにより解析した。シミュレーターにおいて、トラヒックの発生は実際のネットワーク上のパケットを測定し、NFS に特化したトラヒックモデルとしてモデル化を行なった。また、各階層のプロトコルのモデル化と共に、サーバーの CPU 処理性能やディスク処理性能をも考慮したモデルとしてモデル化を行なった。

目 次

第 1 章

はじめに

近年のネットワークの普及により大規模なクライアント・サーバーシステムが広く構築されている。その一つに分散ファイルシステムがある。ネットワークで繋がれた各端末は、このシステムを用いることによってファイルシステムの物理的な保管場所を意識することなく自由に目的のファイルにアクセスすることができる。さらに、最近では管理面での容易さと、リソース共有によるコスト低下の利点からサーバーの集中化の傾向にある。しかし、このような集中化したサーバーによるクライアント・サーバーシステムのネットワーク構成においては、ネットワーク構成要素間の相互の依存関係が非常に強く、ネットワーク構成要素の各々の影響がネットワーク全体の系に大きく影響すると考えられる。例えば、何らかの原因によるファイルサーバーの処理能力の低下が、再送、輻輳をまねき、システム全体のサービス品質の低下を引き起こす。そして、このような影響が考えられるにもかかわらず、実際のネットワークの構築においては、設計者の経験的なものに頼られている。

本研究では、ネットワーク構成要素の各々の部分的な変動がネットワーク全体の系にどのように影響を及ぼすかを解析し、それぞれの最適な構成を求めることを目的とする。

これまでのシステム性能評価手法としては、実測、解析的な求解、シミュレーションがあげられる。実測は、評価対象を実際に測定し評価・解析するものであり、確実性が高い。しかし、実際に評価対象が存在していなければならないため、事後の解析・評価となる。解析的な求解は、評価対象を確率モデルを用いてモデル化する。そして、待ち行列理論を用いて解析・評価を行なう。しかし、評価対象が複雑化してくると同時にそのモデルも複雑化し、その解析が困難となる。シミュレーションは、評価対象のふるまいを忠実に再現

し、それらを動作させて解析・評価を行なうものである。

本論文では、性能評価の一手法としてシミュレーターを作成し、シミュレーションによってネットワークトラヒック特性の解析を行なう。まず、実際の Ethernet 上のトラヒックを計測し、NFS に特化したトラヒック発生のモデル化を行なう。次に、各階層のプロトコルの動作を詳細に記述したネットワークモデルと共に、サーバーの機械的な性能も考慮した NFS サーバモデルとして、シミュレーターを作成する。そして、ネットワーク構成要素の各々の変動がネットワーク全体の系にどのように影響していくかを解析していく。

以下に、本論文の構成を述べる。第二章では分散ファイルシステムである NFS について解説し、NFS の特徴について述べる。第三章では実測値による NFS に特化したトラヒック発生モデルの手法と本論文で開発したシミュレーターについて述べる。第四章では開発したシミュレーターを用いてシミュレーション実験を行ない、出力結果の考察を行なう。最後に第五章でで結論を述べる。

第 2 章

NFS(Network File System)

この章では分散ファイルシステムの一つである NFS(Network File System) について述べる。

NFS は Sun Microsystems 社により開発された。NFS はネットワークで接続されたクライアントに、サーバー上のファイルおよびファイル・システムに対する透過的な共有ファイルアクセスを提供する。Sun Microsystems 社はオープンシステムの思想を実践し、異なったメーカーの異なったオペレーティングシステムが動作する端末を接続できるように設計されている。また、NFS のプロトコルの仕様は公開されており、その資料も安く、誰でも入手できる。そのため NFS は分散ファイルシステムの事実上の標準になっている。

2.1 NFS のプロトコル階層

図 2.1 に NFS プロトコル階層構造を示す。NFS はメッセージを転送するための TCP/UDP と、リモートプロシジャーコール (RPC:Remote Procedure Call) 機構、そして外部データ表現 (XDR:External Data Representation) から成る。RPC はクライアントとサーバー間の通信機構を提供し、XDR はデーター表現においてマシンごとの機種依存を吸収する。

プロトコルの各層はそれぞれ独立しており、他の部分の層に何の影響も与えない。一方の端末のある層のプロトコルは相手側の端末の同層のプロトコルと通信を行なう。通信を行なう情報の単位はそれぞれ層ごとに定義されている。

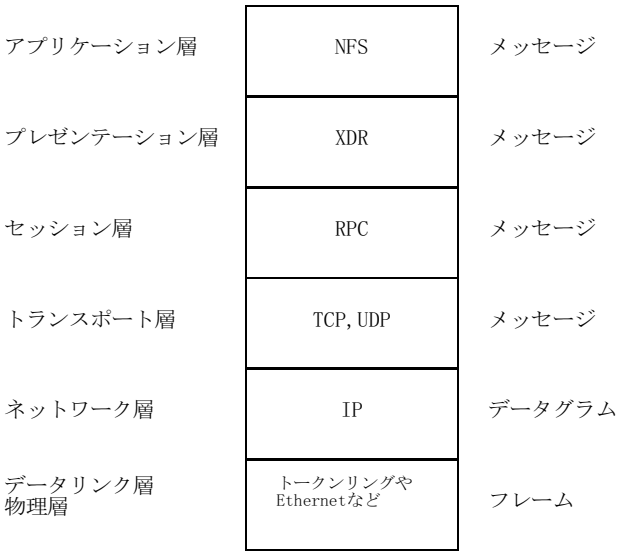

図 2.1: NFS のプロトコル階層と各層で交換される情報の単位

2.1.1 RPC(Remote Procedure Call) 機構

NFS はRPC(Remote Procedure Call:遠隔手続き呼び出し) の上に実現されている。NFS はRPC を介することによって、ネットワーク・プログラミングのほとんどを省くことができる。

RPC の動作は非常に単純で、その使用法はローカルなプロシージャコールと同様である。すべてのメッセージは要求か応答のどちらかで、その通信は同期的に処理される。具体的な動作を図 2.2 に示す。クライアント側のプロセスから送られた要求は、サーバーで受理され、要求を処理し、返答を返す。その間、要求を送ったクライアント側のプロセスはサーバーから返答を受けとるまでブロックされる。

2.2 NFS の構成

NFS は、クライアント・アプリケーションからの透過的なファイル・アクセスを可能にするために様々な特徴を持つ。以下、その構成を簡単に示す。

2.2.1 NFS のプロシージャーコール

NFS サーバーは 15 種類のプロシージャコールを提供している。以下、それらについて解説する。

1. NFSPROC_GETATTR

   ファイルのタイプ（通常ファイル、ディレクトリなど)、許可、サイズ、所有者、最終アクセス時刻などのファイル属性を返す。

2. NFSPROC_SETATTR

   ファイルの属性を設定する。許可、所有者、グループ所有者、サイズ、最終アクセス時刻、最終修正時刻など、ファイル属性のサブセットのみが設定できる。

3. NFSPROC_STATFS

   利用可能なハード容量、転送最適サイズなど、ファイルシステムの状況を返す。

4. NFSPROC_LOOKUP

   ファイル検索。ユーザー・プロセスが NFS サーバー上のファイルをオープンするごとに、クライアントによって呼び出されるプロシジャー。ファイル・ハンドルがファイル属性とともに返される。

5. NFSPROC_READ

   ファイルから読み込み。クライアントは、ファイル・ハンドル、先頭バイト・オフセット、読み込み最大バイト数 (最大 8192 バイト) を指定する

6. NFSPROC_WRITE

   ファイルへの書き込み。クライアントは、ファイル・ハンドル、先頭バイト・オフセット、書き込みバイト数、書き込みデーターを指定する。NFS の書き込みは同期が必要になる。サーバーはデーター (および更新情報) のディスクへの書き込みが成功するまで OK の返答は返さない。

7. NFSPROC_CREATE

   ファイルの作成。ディレクトリにファイルのエントリを確保、登録する。

8. NFSPROC_REMOVE

   ファイルの削除。ディレクトリからファイルのエントリを削除する。

9. NFSPROC_RENAME

   ファイル名の変更。ディレクトリのファイルのエントリの名称部分を変更する

10. NFSPROC_LINK

    ファイルへのハード・リンクを設定する。ハード・リンクはUNIXの概念の1つで、ディスク上の指定されたファイルがそのファイルを指す複数のディレクトリ・エントリを持つことができる。

11. NFSPROC_SYMLINK

    ファイルへのシンボリック・リンクを作成する。シンボリック・リンクは、他のファイルの名前を含むファイルである。シンボリック・リンクを参照するほとんどの操作は、実際にはシンボリック・ファイルによって示されるファイルを参照する。

12. NFSPROC_READLINK

    シンボリック・リンクを読み、それが示すファイル名を返す。

13. NFSPROC_MKDIR

    ディレクトリを作成する。

14. NFSPROC_RMDIR

    ディレクトリを削除する。

15. NFSPROC_READDIR

    ディレクトリを読み込む。

2.2.2 NFS ファイルシステムのマウント

NFSの各クライアントは、NFSサーバー上のファイルにアクセスするため、NFSサーバー上のファイルツリーをローカルのファイルツリー上にマウントする。これらはMOUNT

プロトコルを用いて、クライアント側によって行なわれる。一旦マウントが行なわれると、通常のファイルシステムと同様にアクセスすることが可能となる。

NFS クライアントは NFS サーバーマウント時に様々なオプションを設定することができる。以下に性能解析において重要なものを示す。

hard soft オプション

このオプションはクライアントの要求の再送アルゴリズムを決定する。もし何らかの原因でサーバーのサービス品質が低下し、サーバーからの応答が規定時間を待っても返ってこなかった場合、クライアントは先ほどと同じ要求を再度送信する。

soft マウントされている場合、この再送を規定回数だけ行ない、サーバーがなおも応答を返してこないなら、アプリケーションプログラムにエラーを返す。hard マウントされている場合、サーバーが応答を返してくるまで再送を繰り返す。重要な書き込み動作を行なう場合には hard マウントが使われる。標準ではすべての NFS ファイルシステムは hard マウントされる。

timeo オプション

クライアントの要求に対して、サーバーが返答するまでのタイムアウト時間を規定する。標準ではこのタイムアウト時間は 0.7 秒となる。

2.2.3 仮想ファイル・システム (VFS:Virtual File System)

クライアントのアプリケーションは仮想ファイルシステム (VFS:Virtual File System) を用いることによって、NFS サーバー上のファイルを透過的にアクセスすることができる。例えば図 2.3において、ユーザー・プロセスからのファイルアクセス要求は、VFS によってローカルなファイルへのアクセス要求であるか、NFS サーバー上のファイルへのアクセス要求であるかを判断され、適切に処理が渡される。

このように VFS は様々なファイルシステムの相違点を隠し、一貫性のあるインタフェースを提供する。

2.2.4 nfsd デーモン

NFS サーバーはクライアントから送られてくる要求の対応に NFS デーモン (nfsd) を使用する。クライアントから来る複数の要求を並列的に実行するため、NFS サーバー内ではプロセスを用いて複数の NFS デーモンが起動されている。そして、各 NFS 要求に対して疑似ラウンドロビン式で処理を行なっていく。つまり、1 つの要求への対応が終了すると、待ち行列の最後尾で次の要求の到着を待つ。標準ではこの NFS サーバーは 8 つ起動されている。

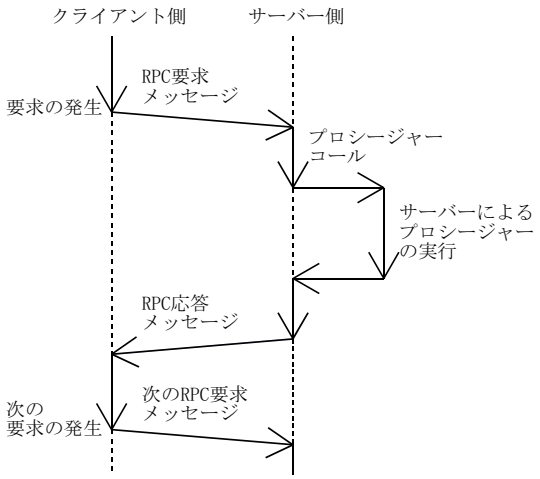

図 2.2: RPC の動作

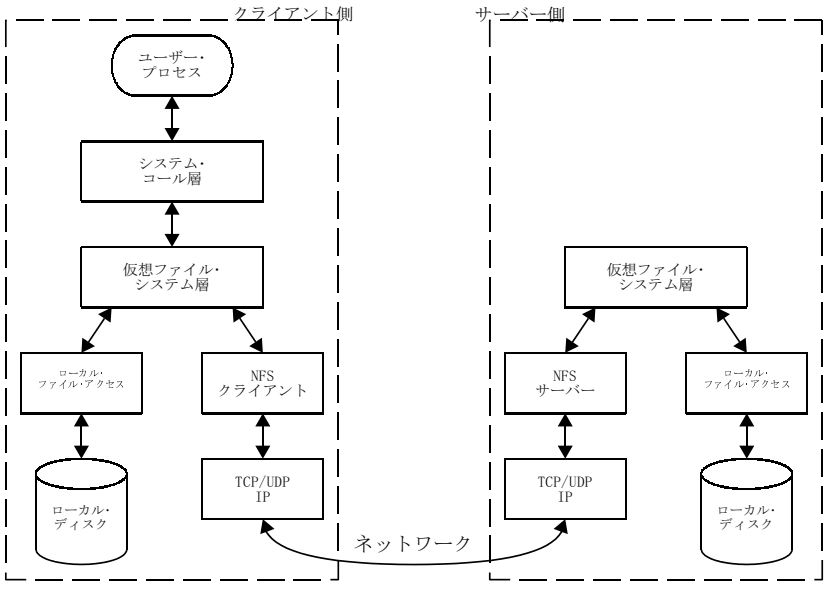

図 2.3: クライアント・サーバーシステムによる NFS の構成図

第 3 章

シミュレーターの概要

この章では本研究で開発したシミュレーターについて述べる。

図 3.1 にシミュレーターのモデルの全体構成を示す。本シミュレーターは以下に示す項目についてモデル化を行なった。

- ユーザートラヒックモデル
  - タイムアウト時間
  - hard マウント（最大再送回数は無限大）
- サーバーモデル
  - プロセス
  - CPU 処理時間
  - ディスクアクセス時間
- ネットワークモデル
  - フラグメント・リアセンブリ
  - ルーティング
  - CSMA/CD 方式
  - 台形型バイナリ・エクスポネンシャル・バックオフ

各部は大きく分けてユーザートラヒックモデル、NFS サーバーモデル、ネットワークモデルの３つから構成される。様々な負荷変動によるトラヒック特性を評価するため、実際の動作を詳細にモデル化した。また、それぞれはモジュール化されており、別のモデルと置き換えることによって、様々なトラヒック特性の評価を行なうことができる。

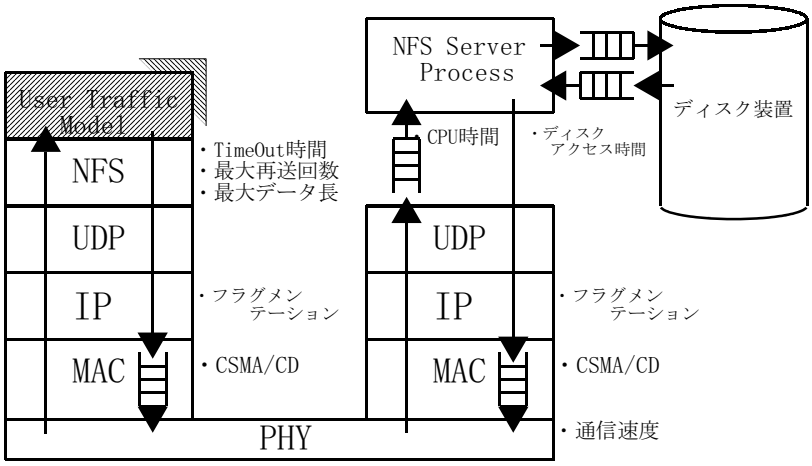

図 3.1: 待合わせシステム

3.1 トラヒックのモデル化

トラヒックのモデル化は各クライアントがパケットをどのように発生しているかをモデル化したものである。そして、シミュレーション結果に影響する非常に重要な部分である。

従来のトラヒック発生の多くは、全てのクライアントを総合し、そのトラヒックの発生をランダムであると仮定し、その到着間隔が指数分布に従うようにモデル化を行なっていた（ポアソン到着)。しかしながら、NFS のトラヒック発生は非常に複雑であり、従来の方法では対応できないと考えられる。そこで、NFS のトラヒック発生を分析し、単一クライアントのトラヒック発生を対象としたモデル化を行ない、それを多重化することによってトラヒック発生モデルとする。

NFS サーバーはクライアント側のユーザープロセスに対して 15 種類のプロシージャー

```
886955162.508618 eve.1eb > is24e1s01.nfs: 100 getattr [|nfs]
886955162.509468 is24e1s01.nfs > eve.1eb: reply ok 96 getattr DIR 40777 ids 0/30 sz 512
886955162.510789 eve.1ec > is24e1s01.nfs: 100 getattr [|nfs]
886955162.511562 is24e1s01.nfs > eve.1ec: reply ok 96 getattr DIR 40755 ids 11023/1000 sz 512
886955162.512798 eve.1ed > is24e1s01.nfs: 100 getattr [|nfs]
886955162.514026 is24e1s01.nfs > eve.1ed: reply ok 96 getattr DIR 40755 ids 11023/1000 sz 512
886955162.515239 eve.1ee > is24e1s01.nfs: 100 getattr [|nfs]
886955162.515929 is24e1s01.nfs > eve.1ee: reply ok 96 getattr REG 100644 ids 11023/1000 sz 148
886955162.517603 eve.1ef > is24e1s01.nfs: 100 getattr [|nfs]
886955162.518279 is24e1s01.nfs > eve.1ef: reply ok 96 getattr REG 100644 ids 11023/1000 sz 148
886955162.522293 eve.1f0 > is24e1s01.nfs: 112 read [|nfs]
886955162.523230 is24e1s01.nfs > eve.1f0: reply ok 248 read
```

図 3.2: cat コマンドによる NFS オペレーション

コールを提供している (2.2.1参照)。そして、ユーザープロセスはユーザーからの命令を15 種類のプロシージャーコールを組み合わせ、複数の NFS 要求を NFS サーバーへ送ることによって処理を完了する。NFS 要求の処理は 2.1.1で述べたように同期式であり、このため要求と次の要求の発生には依存関係が存在する。

図 3.2 に NFS クライアント（クライアント名:eve）の/mnt にマウントされている NFS サーバー（サーバー名:is24e1s01）上のファイルに対して「`cat /mnt/exp/1-1/simulate.out`」を実行した場合のクライアントとサーバー間のやりとりを示す。

ここでは、ユーザーからのキー操作による命令を処理するため、アプリケーションは 6 つの NFS 要求を NFS サーバーへ送ることによって処理を完了している。これからもわかる通り、クライアントからの NFS 要求に対して、サーバーが応答を返す同期式の処理が行なわれている。はじめの 10 行までの getattr はディレクトリとファイルの検索のために発行され、最後の read によって目的のファイルの読み出しが行なわれる。そしてアプリケーションはユーザーからのキー操作による命令の処理が完了するとユーザーに結果を示し、処理の完了を知らせる。その後、ユーザーは次のキー操作を行なう。

このように NFS 要求の発生には 2 種類の発生要因が存在する。一つはユーザーのキー操作による命令の要求、つまり、アプリケーションが最初の NFS 要求を発生するタイミングである。もう一つはユーザーの要求を処理するための一連の NFS 要求を発生するタイミングである。前者は、ユーザーがアプリケーションの出力結果を見て、考え、次の命

令をアプリケーションに要求するまでの時間間隔であると考えられる。また後者は、RPCが同期式の処理を行なうため、クライアントが送った NFS 要求の返答までの時間間隔、言い替えれば、この時間間隔はクライアントがサーバーへ送った NFS 要求の種類に大きく依存しており、クライアントがサーバーへ送る NFS 要求の発生順序であると考えられる。

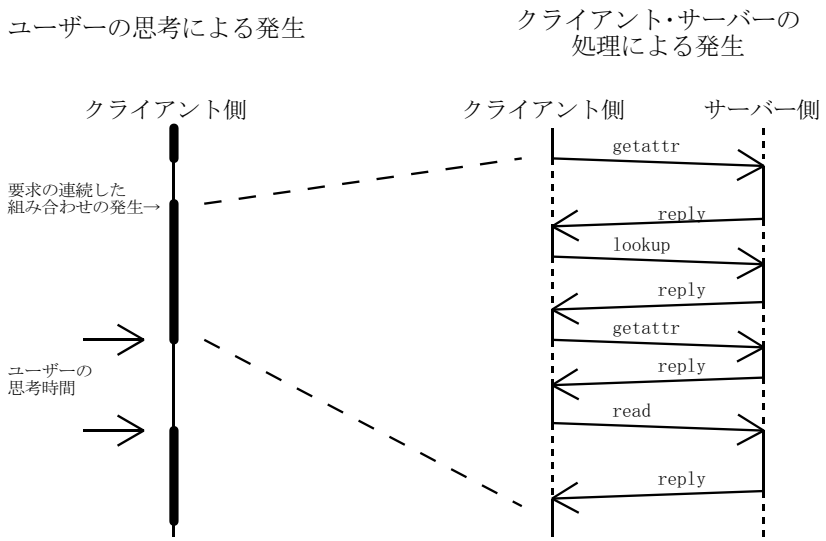

図 3.3: トラヒックの発生

そこで本シミュレーターのトラヒック発生部においては、ユーザーの思考時間間隔と、NFS 要求の発生順序に着目してモデル化を行なった。そして、モデルのパラメーターの決定には実際に稼働している LAN のパケットを測定し、それに基づいて決定した。

3.1.1 ユーザートラヒックモデル

パケットの測定は本学情報科学研究科計算機アーキテクチャ講座内の LAN を対象とした。講座内の各端末それぞれに対して UNIX の tcpdump コマンドを用いて NFS パケットの発生時間と NFS 要求の種類について測定を行なった。測定期間は 1997.11.7〜1997.12.7 まで行なった。この測定結果からユーザーの思考によるキー操作に依存した部分とクライアント・サーバー間の処理に依存した部分とに分ける。

一般に人間のキー入力のスピードは 0.7 秒であると言われている。そこで、隣り合う要求の発生間隔時間が 0.7 秒以上である場合、その間隔はアプリケーションの処理終了によって次の処理を与えるためのユーザーの思考時間であると仮定した。そして、それによって分割された NFS 要求の一連の組をアプリケーションによって発行された NFS 要求群であると仮定した。

ユーザーの思考によるキー操作に依存した部分

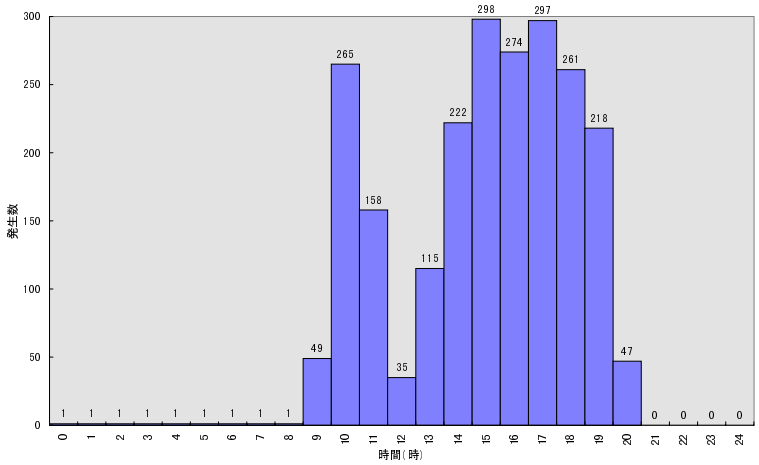

図 3.4: あるユーザーが発生する NFS 要求群の時間的分布

図 3.4は、あるユーザーによってキー操作が行なわれたと思われる回数の時間的分布を示したグラフである。ユーザーによるキー操作はその時の気分や状態によって変化すると考えられる。また、端末に座っていない場合においてはキー操作は一切行なわれない。このような特定の状態や環境による挙動をデーターから排除するため、以下の方法で 3 つのクラスに分けた。

1. 全クライアントの 1 時間毎の発生数を出す。
2. 1 時間毎の発生数によってクラス分けを行なう。
   - クラス 1 … 発生数が 3150〜3850 のもの

表 3.1: 各モデルの平均と分散

	平均 (μ)	分散 (σ^2)
モデル 1	0.809669	0.005346
モデル 2	3.278112	2.949965
モデル 3	17.503021	1043.761295

- クラス 2 … 発生数が 900〜1100 のもの
- クラス 3 … 発生数が 180〜220 のもの

そして、各クラスにおける思考時間間隔 t の分布を 3.1式によって与えられる密度関数 f(t) のパラメータとして抽出し、モデル化を行なった。測定時において思考間隔時間を 0.7 秒以上と仮定したので、3.1式は思考時間間隔 t が 0.7 秒未満を 0 とした正規分布 $N(\mu, \sigma^2)$ である。

$$f(t) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \int_{0.7}^{\infty} e^{\frac{-(t-\mu)}{2\sigma}} dt & t \geq 0.7 \\ 0 & t < 0.7 \end{cases} \tag{3.1}$$

表 3.1に各々のクラスの平均と分散を示す。そして、図 3.5, 図 3.6, 図 3.7は実際の測定結果と計算による分布を示す。これらモデルは平均と分散との値から、モデル 1 は平均 0.81 秒で分散 0.005 秒であり、平均がトラヒック測定時に仮定した人間のキー入力スピードと言われている 0.7 秒に近く、また分散の値からあまりばらつきがみられない。これは、ユーザーが端末に対して特定の操作を繰り返し行なっている場合か、非常に集中して端末の操作を行なっている場合のモデルであると考えられる。モデル 2 は平均 3.28 秒と分散 2.95 秒から、普通に端末に向かって操作を行なっている場合のモデルであると考えられる。モデル 3 は平均 17.50 秒で分散 1043 秒であり、平均も長く、分散においては非常にばらついている。これは、端末の操作のみを行なっているのではなく、他にも端末の操作以外の作業をしているような場合のモデルであると考えられる。

クライアント・サーバー間の処理に依存した部分

アプリケーションによって発行される NFS 要求の発生は、その要求と応答のメッセージ交換が同期的に行なわれているため、サーバーの応答メッセージの到着に依存してい

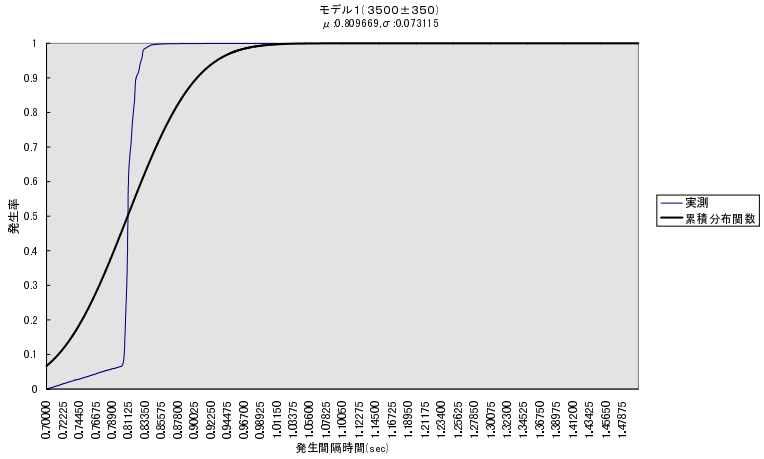

図 3.5: モデル 1

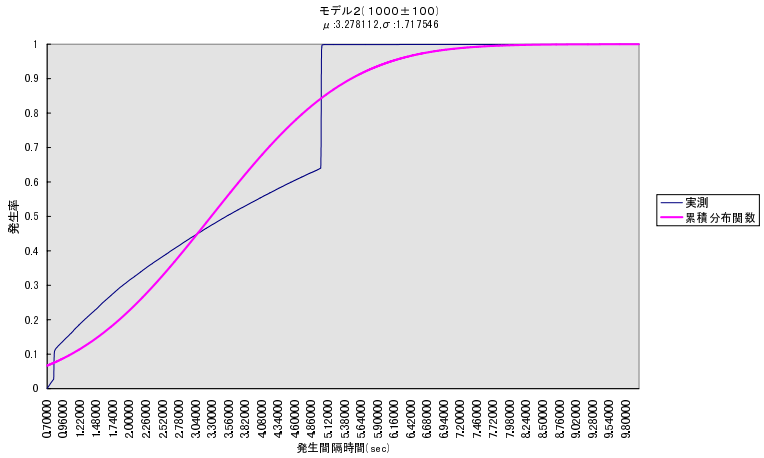

図 3.6: モデル 2

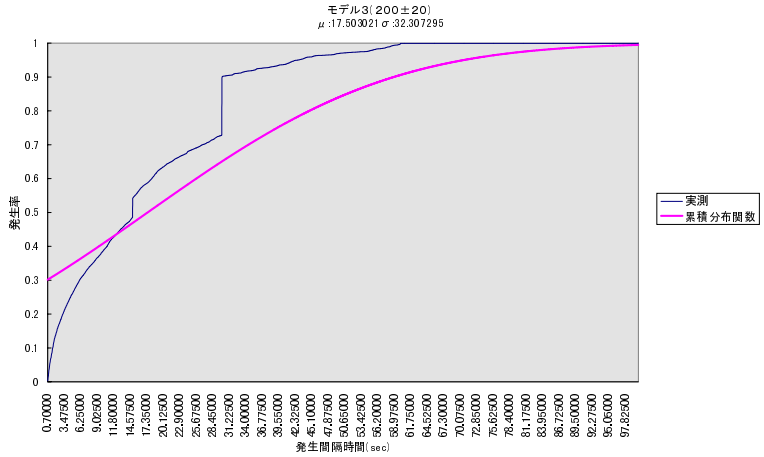

図 3.7: モデル 3

る。そこで、本モデルにおいては、NFS 要求がどのような順序で発生されたかについて着目した。

NFS は RPC を用いて 15 種類のプロシージャーコールを提供する。この 15 種類のプロシージャーコールにおいて、NFS 要求群それぞれの発生順序を集計し、その発生順序を状態遷移確率として表した。表 3.2はあるクライアントが発生した NFS 要求の発生順序を状態遷移確率として表に表したものである。そして、表 3.2の状態遷移確率を基に状態遷移図で表したものを図 3.8に示す。

具体的な方法は、

1. 測定結果を各クライアント毎に分ける。
2. 要求と要求との発生間隔が 0.7 秒のものを分割点として NFS 要求の系列を抽出する。
3. 系列内での要求から要求への遷移数を集計する。
4. 集計結果から、NFS 要求の発生順序を示す確率遷移行列を算出する。
5. 全てのクライアントの確率遷移行列を用いて平均化し、NFS 要求発生順序のモデルを作成する。

表 3.3に計測結果から算出した確率遷移行列を示す。

最後に、ユーザーネットワークモデルの動作の流れを示す。

1. 正規分布に従う乱数によって決定された思考時間の空白の後、NFS 要求群発生部へ処理が移行する。

2. NFS 要求群の発生部では、確率遷移行列を用いて 15 種類の NFS プロシージャーコールをサーバーモデルに向けて発行する。

3. NFS 要求の各発行間隔時間はサーバーモデルからの返答によって決まる。

4. ここでもしサーバーモデルからの返答が規定時間（$timeout = 0.7sec$）を過ぎてしまった場合は、再送が行なわれる。この再送は無限回続く。

3.2 NFS のモデル化

NFS ファイルサーバーの性能には、CPU、キャッシュ、メモリ、ディスクとさまざまな要素が絡み合っている。しかし、本研究の目的はネットワークトラヒック特性の解析であり、ハードウェアの性能を厳密に解析することではない。従って、モデル化に際しては CPU 処理時間とディスクアクセス時間という大まかな抽象化の観点からサーバーサブモデル、ディスクサブモデルという２つのモデル化を行なった。

3.2.1 サーバーサブモデル

サーバーサブモデルではプロセス機構と CPU 処理時間についてモデル化を行なった。

NFS サーバーでは、クライアントからの NFS 要求を、プロセスを用いて複数のデーモンにより対応することは 2.2.4で述べた。本シミュレーターでも実際と同様にこれらのプロセス機構をモデルに記述した。

NFS 要求を受けとったデーモンはその NFS 要求を処理するため、所定の時間 CPU を占有する。当然、CPU の占有時間は NFS 要求の種類によって異なる。そして、必要ならばディスクへアクセス要求を出し、ディスクからの返答が帰ってくるまでデーモンはブロックされる。NFS 要求の処理が完了すると、デーモンは適当な大きさの NFS 返答メッセージをクライアントへ返す。本シミュレーターのサーバーサブモデルは各 NFS 要求に

表 3.2: あるクライアントの NFS 要求発生の状態遷移確率

遷移前	遷移後																	
	getattr	setattr	statfs	lookup	read	write	create	remove	rename	link	symlink	readlink	mkdir	rmdir	readdir	null	end
start	0.927424	0.000000	0.000000	0.045491	0.005576	0.006121	0.000042	0.000000	0.000210	0.000000	0.000000	0.013626	0.000000	0.000000	0.000000	0.001509	0.000000
getattr	0.355872	0.005799	0.008490	0.205955	0.125654	0.007337	0.001307	0.000319	0.000099	0.000000	0.000000	0.040585	0.000000	0.000000	0.080049	0.008062	0.160472
setattr	0.135572	0.000000	0.000000	0.090796	0.018657	0.667910	0.000000	0.007463	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.079602
statfs	0.000000	0.000000	0.000000	0.998706	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001294	0.000000	0.000000	0.000000	0.000000	0.000000
lookup	0.228184	0.002332	0.000000	0.613604	0.008117	0.000984	0.010619	0.006302	0.001993	0.000365	0.000000	0.065171	0.000008	0.000000	0.000755	0.000246	0.061321
read	0.046510	0.000000	0.000000	0.169614	0.718197	0.020531	0.000413	0.000000	0.000000	0.000000	0.000000	0.019479	0.000000	0.000000	0.000124	0.000041	0.025091
write	0.050305	0.000000	0.000000	0.016243	0.044523	0.870479	0.001945	0.000000	0.000000	0.000000	0.000000	0.000105	0.000000	0.000000	0.000000	0.000000	0.016400
create	0.995804	0.000000	0.000000	0.001399	0.000699	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000699	0.000000	0.000000	0.000000	0.000000	0.001399
remove	0.584383	0.000000	0.000000	0.224181	0.016373	0.000000	0.000000	0.018892	0.000000	0.000000	0.000000	0.039043	0.000000	0.000000	0.000000	0.000000	0.117128
rename	0.613281	0.000000	0.000000	0.000000	0.382813	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.003906
link	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
symlink	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
readlink	0.114651	0.000000	0.000000	0.863062	0.000842	0.000077	0.000077	0.000077	0.000000	0.000000	0.000000	0.019913	0.000000	0.000000	0.001225	0.000000	0.000077
mkdir	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
rmdir	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
readdir	0.113700	0.000058	0.000000	0.285415	0.008271	0.000058	0.000000	0.000000	0.000408	0.000000	0.000000	0.006640	0.000000	0.000000	0.569024	0.000000	0.016426
null	0.928224	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.025547	0.046229

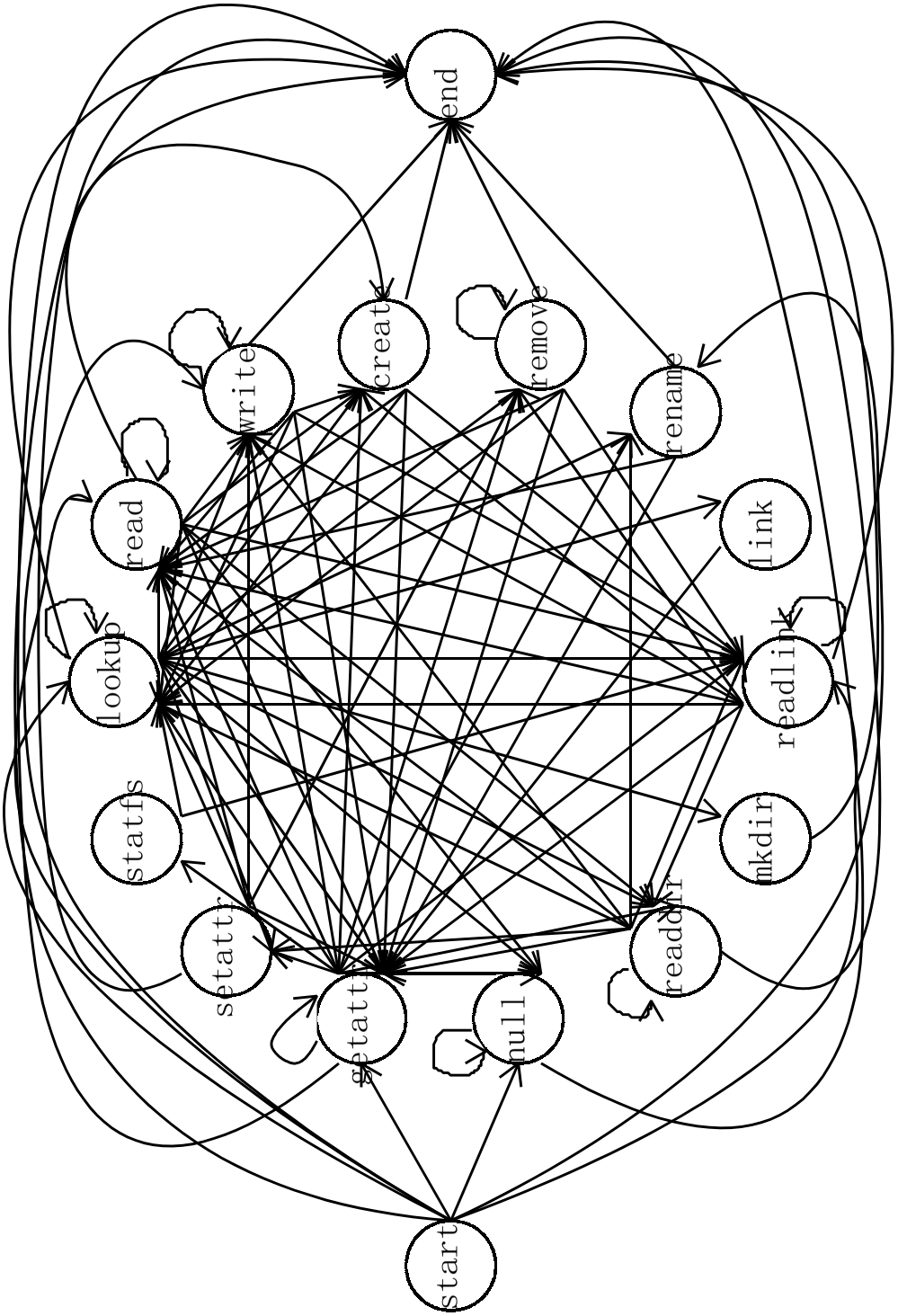

図 3.8: あるクライアントの NFS 要求発生の状態遷移図

表 3.3: 確率遷移行列

遷移前	遷移後																
	getattr	setattr	statfs	lookup	read	write	create	remove	rename	link	symlink	readlink	mkdir	rmdir	readdir	null	end
start	0.717392	0.000019	0.000004	0.186323	0.014448	0.045696	0.000618	0.000332	0.000034	0.000000	0.000001	0.034167	0.000001	0.000000	0.000036	0.000930	0.000000
getattr	0.354226	0.014170	0.000813	0.225292	0.074838	0.025399	0.001458	0.000372	0.000027	0.000135	0.000000	0.043534	0.000000	0.000000	0.043924	0.000326	0.215486
setattr	0.136342	0.001690	0.000000	0.075019	0.086408	0.630260	0.000032	0.015655	0.000000	0.000000	0.000000	0.002243	0.000000	0.000000	0.000663	0.000000	0.051687
statfs	0.130412	0.000000	0.320466	0.480786	0.003478	0.000174	0.000696	0.001217	0.000000	0.000000	0.000000	0.011824	0.000000	0.000000	0.000174	0.000174	0.050600
lookup	0.222830	0.001050	0.000080	0.605766	0.013055	0.005134	0.016950	0.012425	0.001285	0.000469	0.000149	0.061002	0.000103	0.000064	0.002967	0.000152	0.056520
read	0.040127	0.000037	0.000009	0.107728	0.765366	0.022676	0.000825	0.000069	0.000004	0.000000	0.000000	0.023641	0.000000	0.000000	0.002673	0.000334	0.036513
write	0.131055	0.000299	0.000001	0.040763	0.025813	0.716031	0.002397	0.000165	0.000022	0.000000	0.000000	0.001238	0.000000	0.000000	0.000084	0.000030	0.082101
create	0.992365	0.000020	0.000000	0.002108	0.002483	0.001369	0.000069	0.000079	0.000010	0.000000	0.000000	0.000374	0.000000	0.000000	0.000552	0.000000	0.000571
remove	0.438706	0.000057	0.000000	0.348620	0.044582	0.071099	0.000057	0.008863	0.000043	0.000000	0.000000	0.012720	0.000000	0.000085	0.000369	0.000000	0.074800
rename	0.689602	0.000565	0.000000	0.052840	0.089715	0.013281	0.000989	0.003815	0.000000	0.000000	0.000000	0.009749	0.000000	0.000000	0.001554	0.000000	0.137892
link	0.347669	0.000000	0.000000	0.407420	0.000000	0.000657	0.141825	0.100460	0.000000	0.000328	0.000000	0.000000	0.000000	0.000000	0.000985	0.000000	0.000657
symlink	0.839823	0.000000	0.000000	0.090602	0.055296	0.006490	0.000000	0.000000	0.000000	0.000000	0.000779	0.003894	0.000000	0.000000	0.000260	0.000000	0.002856
readlink	0.063628	0.000009	0.000003	0.857315	0.010611	0.000146	0.000036	0.000050	0.000000	0.000000	0.004748	0.058828	0.000000	0.000000	0.004021	0.000146	0.000457
mkdir	0.677596	0.000000	0.000000	0.245902	0.001821	0.001821	0.000000	0.000000	0.000000	0.000000	0.000000	0.005464	0.000000	0.000000	0.000000	0.000000	0.067395
rmdir	0.428571	0.000000	0.000000	0.477922	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.002597	0.000000	0.090909
readdir	0.114617	0.000080	0.000373	0.235123	0.017863	0.000625	0.000082	0.000078	0.000082	0.000002	0.000000	0.023330	0.000002	0.000080	0.588704	0.000023	0.018938
null	0.458405	0.000000	0.000000	0.023622	0.014207	0.003081	0.000000	0.000000	0.000000	0.000000	0.000000	0.000856	0.000000	0.000000	0.000514	0.291852	0.207463

おける CPU 処理時間、ディスクアクセスを行なうか否か、返答パケットの大きさのパラメーターを持つ。

CPU 処理時間のパラメーターの抽出

各パラメーターは実測データーから抽出した。対象とした NFS サーバーは富士通 S-4/CL とした。

以下に各 NFS 要求の CPU 処理時間の測定方法を示す。

1. クライアントとサーバーが各 1 台から成るネットワークを構築する。これによって、Ethernet の衝突の影響を無視できる。
2. クライアントから適当なアプリケーションを動作させ、その応答時間とパケット長を tcpdump コマンドによって計測する。
3. サーバー側では NFS 要求を並列に処理させないために、サーバー内のデーモン (nfsd) を 1 つにした。
4. 各 NFS 要求のそれぞれの応答時間を計測した。

次に、パラメーター決定での基準を以下に示す。

1. read と write のみは必ずディスクへアクセス要求を出すとし、その CPU 処理時間は read と write の測定結果の最小値を用いた。
2. 測定結果から発生頻度が高く、応答時間のばらつきが大きいものに関しては、ディスクへのアクセスをある確率で行なわせ、CPU 処理時間を測定結果の最小値を用いた。ディスクアクセスを行なう確率は、以下の 3.2式を用いて応答時間の平均値と最小値より算出する。
3. NFS 返答のパケット長は各 NFS 返答毎の実測データを平均することによって決定した。
4. null は測定できなかったので、他の要求内で、最小のデータ長、最小の平均応答時間を用いた。

5. read と write のデータ長はユーザーがアクセスするファイルによって異なるが、ここでは UDP の最大パケット長 8kbyte の固定長とした。

$$ProbabilityOfDiskAccess = \frac{AverageResponse - MinimalResponse}{AverageDiskAccess} \tag{3.2}$$

表 3.4に本シミュレーションで用いた各 NFS 要求の CPU 処理時間を示す。

3.2.2 ディスクサブモデル

ディスクサブモデルは各プロセスから発生されたディスクへの要求のアクセス処理時間をモデル化している。ディスクへのアクセス時間はディスク装置のアームの動作スケジューリングと、ディスク装置の機械的な性能によって影響する。

ディスク・ドライバーのアーム動作スケジューリングには３つの方法がある。第一の方法は到着順サービス (FCFS,First-Come First-Served) である。これは、ディスク・ドライバーが 1 つづつ要求を受け付け、順番に処理を行なっていく方法である。第二の方法は最短シーク順 (shortest seek first、SSF) である。これは、現在のアーム位置からアームを移動させる距離がもっとも短い要求を常に処理していく方法である。第三の方法はエレベーター・アルゴリズム (elevator algorithm) である。これは、まずアームの移動する方向を決める、その方向への目立った要求がなくなるまでアームを移動し、その後アームの移動方向を切り替えるという方法である。本モデル化は、ディスク装置のアーム動作スケジューリングに第一の方法である FCFS を用いることによって、アームのスケジューリングは行なわない。

次にディスク装置の機械的な構造による影響について述べる。ディスク装置は複数の円盤が組み合わさって構成されている。各円盤において、トラックと呼ばれる複数の同心円に分割され、さらにそのなかで複数のセクターに分割されている。ディスクはこのセクター単位で扱われる。

セクターの読み書きを行うため、各円盤に読み書きヘッドを持つアームが取り付けられている。各円盤のアームは互いに連結されており、一斉に連動して動作する。このため、各円盤のアームはすべて同じトラックに位置する。この各円盤の同一の位置にあるトラックをまとめてシリンダと呼ぶ。

表 3.4: サーバーサブモデルのパラメーター（富士通 S-4/CL）

	実測データ				パラメータ			
NFS 要求	発生率	応答時間 (μsec)			データ長 (byte)		CPU 処理時間	ディスク
	(%)	最大	最小	平均	要求	応答	(μsec)	アクセス割合
getattr	26.95	36240	1004	2068	104	96	1004	0.068
setattr	0.42	71365	8017	12263	136	96	12263	0
statfs	0.04	45577	1024	1217	104	48	1217	0
lookup	35.63	273879	1307	4016	117	69	1307	0.174
read	12.58	442183	1141	3694	116	8192	1141	1
write	6.57	353728	16997	34011	8192	96	1141	1
create	0.68	196731	16920	30729	155	128	30729	0
remove	0.47	605409	17796	33548	123	28	33548	0
rename	0.05	180696	47506	77545	152	28	77545	0
link	0.02	114633	28890	35741	144	28	35741	0
symlink	0.03	581764	765	45999	160	28	45999	0
readlink	4.30	19467	1106	4389	104	40	1106	0.211
mkdir	0.00	594095	793	123904	144	128	123904	0
rmdir	0.00	62224	22010	36933	115	28	36933	0
readdir	3.26	23985	1059	4571	328	112	1059	0.226
null	0.04	-	-	-	120	28	1217	0

表 3.5: ディスク装置のパラメータ

シリンダ数	2036
トラック数/トラック	14
セクタ数/シリンダ	72
セクタサイズ	512 byte
回転速度	5400 rpm
最大回転待ち時間	11.11 msec
単一シリンダシークタイム	2.3 msec
平均シークタイム	10 msec
フルストローク	18 msec
転送レート	3.22 Mbyte/sec

この様な構造上、ディスクアクセス時間はシリンダからシリンダへのアームの移動時間（これをシーク時間と呼ぶ）とディスクの回転待ち時間とデータ転送にかかる時間を加えた値となる。

$$DiskAccessTime = SeekTime + RotationLatency + TransferTime \tag{3.3}$$

そして、シーク時間は次の式で計算される。

$$SeekTime = \begin{cases} 0 & \text{if } x = 0 \\ a\sqrt{(x-1)} + b(x-1) + c & \text{if } x > 0 \end{cases} \tag{3.4}$$

ここで、x はディスクヘッドが移動するシリンダ数で、a,b,c は単一シリンダシーク、フルストローク、平均シークタイムを満たすように選ばれる定数である [8]。移動先のシリンダ番号は 0 から最大値までの間でランダムに選ばれる。また、回転待ち時間も 0 から最大値までの間でランダムに選ばれる。表 3.5 に本モデルで参考にしたパラメータを示す。これは Conner 社 が公表している 3.5 インチディスク CP30540 のパラメータである。

3.3 ネットワークのモデル化

ネットワークのモデル化は、パケットの交換をモデル化している。本シミュレーターにおいては、各階層のプロトコルの実際の動作を忠実に再現し、パケットの交換を完全にシミュレートしている。

なお、今回のシミュレーションにおいてはシミュレーション対象が NFS であるため、IP と 10Mbit/sec の Ethernet をモデル化した。

3.3.1 IP サブモデル

IP は信頼性のないコネクションレス型のデータグラム転送サービスを行なう層として、IP データグラムのフラグメント・リアセンブリ、そしてルーティングを行なっている。

IP の下層のリンク層では最大転送ユニット (MTU) と呼ばれるフレームサイズの限界値が存在する。IP は送信しようとするデーターグラムを上の層から受け取ると、そのデータグラムをどのローカル・インターフェースに送るべきかを判断し（これをルーティングと呼ぶ)、そのインターフェースに対して MTU を照会する。そして IP は MTU とデータグラムとのサイズを比較し、必要であれば MTU よりも小さなサイズに細分化される。この細分化のことをフラグメントと呼ぶ。

一旦 IP データグラムがフラグメント化されると、宛先に到達するまで再構成されることはない。そして、宛先で全てのフラグメント化されたフレームが揃った所で、再構成され、上位層へ渡される。この再構成のことをリアセンブリと呼ぶ。

本モデルにおいてはシミュレーションの性質上、実際の IP と同様に IP データグラムのフラグメント・リアセンブリ、及びルーティングを行なうようにモデル化を行なった。

フラグメント・リアセンブリ及びルーティングのための IP サブモデルの動作の流れは以下の様になる。

- 送信側のサブモデルは、上位層のモデルからデータグラムを受けとると、宛先を判断して、目的のネットワークへのルーティングを行なう。次に、フラグメントの必要があると、データグラムをフラグメント化し、Ethernet サブモデルの送信待ち行列に追加する。

- 受信側のサブモデルは、下位層から受けとったフレームがフラグメント化されてい

ると、そのフレームが属するエントリーに順次追加され、フレームが全て揃った時点で再構成され、上位層へ渡される。

3.3.2 Ethernet サブモデル

Ethernet サブモデルは Ethernet の動作を詳細にモデル化したものである。

Ethernet は直列バス形 LAN でその転送速度は $10Mbit/sec$ である。回線へのアクセスの制御はすべてのノードに完全に分散されており、衝突検出機構つきの 1-パーシステント CSMA 方式に基づく。

ノードは転送を希望すると、回線をリスンする。回線がビジーであると、ノードはそれが空きになるまで待ち、そうでなければ直ちに転送を行なう。2 つ以上のノードが同時に空きの回線上で転送を開始すると、衝突する。衝突したノードは転送を中止し、ランダム時間だけ待ち、全体の手順をもう一度繰り返す。

本シミュレーションの Ethernet サブモデルはこれらの動作を図 3.9 の状態遷移図に従って動作する。この状態遷移図は各端末それぞれを表したものである。

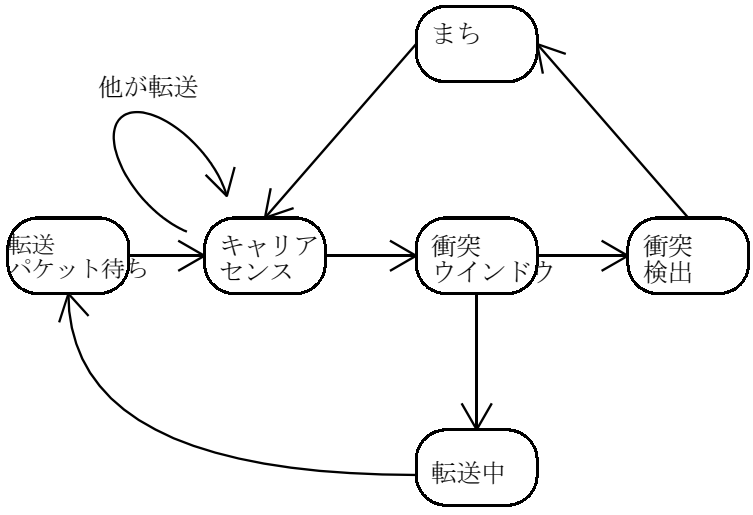

図 3.9: CSMA/CD の状態遷移図

転送パケット待ち 転送するパケットの無い状態である。キューにパケットが入ると、キャ

リアセンスへ遷移する。

キャリアセンス キャリアセンスを行う。回線の状態を逐次検出し、空くまで待つ。空いているなら衝突ウインドウへ遷移する。

衝突ウインドウ 転送を開始する。しかしこの期間に衝突がおこる可能性がある。本シミュレーターではこの期間を $51\mu sec$ とした。所定の期間の経過の後、衝突検出の状態へ遷移する。

衝突検出 衝突の検出を行なう。衝突が起こっていないなら転送中の状態へ遷移する。衝突が起こっているなら衝突待ちの状態へ遷移する。

衝突待ち ランダムな待ちの後、キャリアセンスへ遷移する。この待ち時間間隔は台形型バイナリ・エクスポネンシャル・バックオフのアルゴリズムにより決定される。

転送中 衝突ウインドウで衝突無く転送できた残りのパケットを転送する。この状態において各ノードは回線使用中であることを知っており、衝突が起こることはない。転送が完了すると転送パケット待ちの状態へ遷移する。

衝突ウインドウ

Ethernet の各ノードは、回線へのアクセスの制御を独自に行なっている。今、あるノードが回線の空きを検出し送信を開始したとしよう。このノードが送信を開始したということがネットワーク上のノード全てに行き渡るには一定の時間がかかる。言い替えれば、他のノードの送信を検出するためには信号の伝搬遅延による一定の時間待たないと検出することができない。このことは、チャンネルの空きを検出し送信を開始した後でも最初の一定期間は衝突が発生する可能性があることを意味する。この期間のことを衝突ウインドウ (collision window) と呼ぶ。この値は最悪の場合を考慮したもので、Ethernet の最長パス（4 つのリピータと 2.5km）を収容するため、その値は $51.2\mu sec$ と規定されている。

台形型バイナリ・エクスポネンシャル・バックオフ

衝突が起こった場合の再送処理のための待ち時間を決定するアルゴリズムである。このアルゴリズムは衝突の発生数によって重みづけられた乱数を用いている。そして、他のス

テーション間でアルゴリズムが同期しないようになっている。衝突の発生数による重みづけを用いることによりイーサネットの負化が重くなり衝突の発生が頻繁になると待ち時間間隔が増大することになり、衝突の確率を減らすように働く。

具体的には次式により決定される。

$$\text{再送信間隔 } T = 51.2\mu sec \times n \tag{3.5}$$

整数 n は、次の範囲からランダムに用いられる。

$$0 \leq n < 2^k [k = min(m, 10) m : \text{衝突回数}] \tag{3.6}$$

なお、衝突時の再送は最大 16 回であり、16 回連続で衝突が発生した場合は、上位層へ送信エラーが報告される。

第 4 章

シミュレーション実験

ネットワーク構成要素の各々の部分的な変動がネットワーク全体の系にどのように影響を及ぼすかを分析し、システムの性能を解析するため、本研究で作成したシミュレーターを用いてシミュレーション実験を行なう。

サーバーボトルネック、ネットワークボトルネック時の各部の変動の解析を行なった。

4.1 実験 1

クライアント数の増大によって各構成要素に与える影響を解析する。

4.1.1 方法

シミュレーションは以下の状況を仮定して行なった。単一の $10Mbit/sec$ の Ethernet セグメントにおいて、単一の NFS サーバーに複数台のクライアントが接続された場合を想定する。NFS サーバーのモデル化の対象として富士通 S-4/CL を用いた。シミュレーターの各モデルのパラメータは以下に示す。

- ユーザートラヒックモデル
  - ユーザーの思考時間間隔には表 3.1 内のモデル 1(平均 $\mu = 0.810[sec]$、分散 $\sigma^2 = 0.005[sec]$) を用いた。
  - 要求の発生順序を表す確率遷移行列には表 3.3を用いた。

- NFS 要求のパケット長は表 3.4を用いた。

- サーバーモデル
  - nfsd デーモンの数は 8 個とした。
  - CPU 処理時間、返答パケット長、ディスクアクセス要求の割合には表 3.4を用いた。
  - ディスクのシークタイム ($a = 190, b = 1.97, c = 2300$)、転送レート、回転待ち時間は表 3.5を用いた。

- ネットワークモデル
  - 最大転送ユニット (MTU) は 1500[byte] とした。

この様な状況でクライアント数を変化させた場合のシミュレーションを行なった。

4.1.2 結果と考察

図 4.1 に NFS 要求と NFS 要求群に対する各応答時間を示す。ここで NFS 要求群とは、アプリケーションがユーザーの命令によって発生する複数の NFS 要求を一つの群として考えたものである。例えば、UNIX のコマンド「`ls`」を入力し、応答が帰ってくるまでの時間間隔が要求群応答時間となる。

この図から、クライアント台数が 30 台で要求群の応答時間が 1.27 秒、要求の応答時間が 0.12 秒であり、コマンド入力の応答時間が 1 秒前後であることから、このネットワーク構成における実用領域の限界点であると考えられる。

図 4.2 にサーバー、ディスク、ネットワークの各動作率を示す。この図から、クライアント台数が 20 台でサーバーの動作率の飽和が見られ、サーバーボトルネックであることがわかる。そして飽和点は 80%前後と読みとることができる。

これは、このネットワーク構成における nfsd デーモンのプロセス数が少ないことを意味する。NFS サーバーはクライアントからの NFS 要求を nfsd デーモンプロセスで受け、順次処理を行なっていく。そして、その nfsd デーモンの数は固定数である。さらに、NFS 要求がディスクへのアクセスを伴う場合、nfsd デーモンはディスクからの返答が帰ってくるまでロックされてしまう。つまり、全ての nfsd デーモンがディスクアクセスを伴う NFS

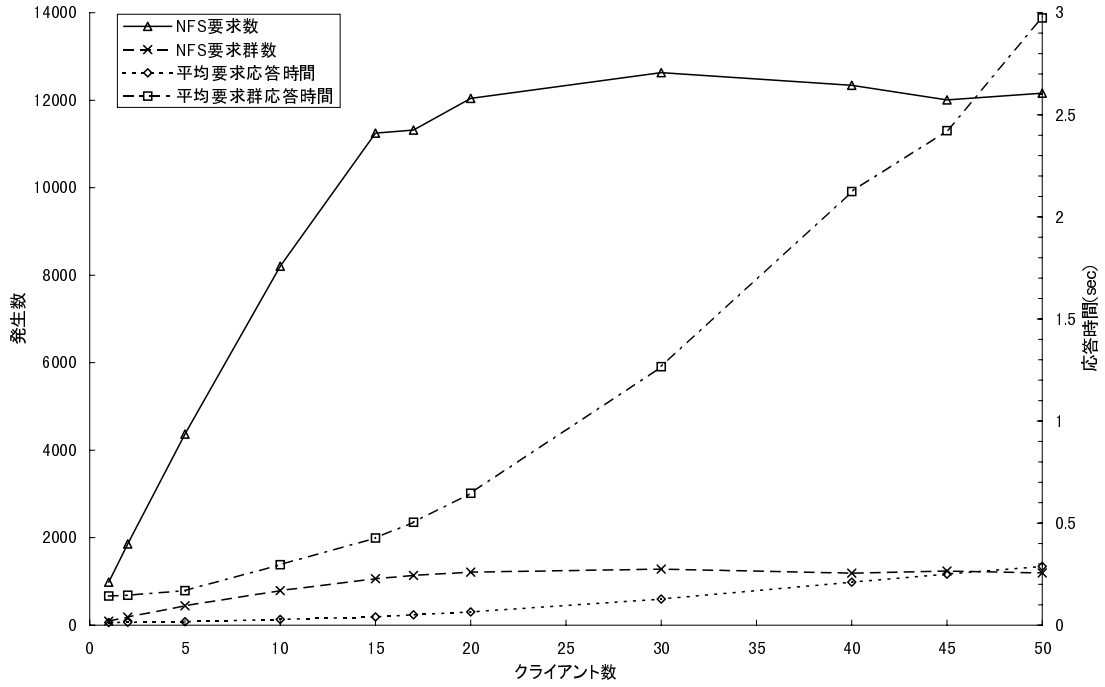

図 4.1: 要求発生数と応答時間

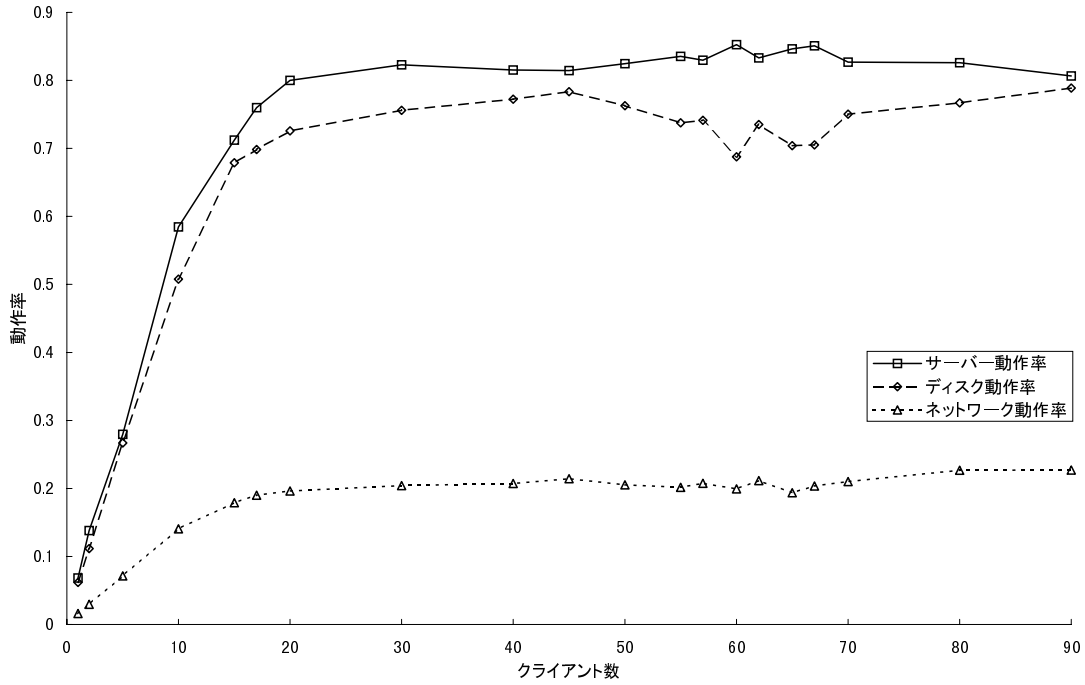

図 4.2: サーバー・ディスク・ネットワークの動作率

要求を受け、ディスクの応答待ちに入り、サーバーの CPU の動作しない状態が 20%であると考えられる。従って、nfsd デーモンのプロセス数を増やし、ディスクアクセスを伴う NFS 要求によって全ての nfsd デーモンがロックされる状況を減らすことによって、飽和点の動作率を上げることができる。

さて、サーバーの飽和以降、クライアント台数が増加しているにもかかわらず、ディスク及びネットワークの動作率の上昇が頭打ちになっている。これはこのシステムの系内で次のような状況が起こっているためであると考えられる。

クライアントからの NFS 要求はサーバーへ送られ処理されるが、サーバーの処理能力以上のものはサーバーの入力キューへ溜っていく。そして順番に処理されていく。このため、サーバーでの処理時間と共にキューでの待ち時間も加わり応答時間が増加する。一方、クライアントの NFS 要求の発生はサーバーからの応答を受けてから次の新しい NFS 要求を発生するという同期式の処理を行なっている。従って、サーバーからの応答時間という形でクライアントの NFS 要求の発生が押えられる。つまり、このシステムでは系内のトラヒック量がサーバーの処理量によって押えられてしまうため飽和以降のディスク、ネットワークの動作率が上昇しないと考えられる。

図 4.3 はシステム系内の NFS 要求数、再送数、そして Ethernet の衝突数と平均応答時間を表したものである。この図から、NFS 要求数、Ethernet の衝突数、共にクライアント台数が 20 台のサーバー動作率飽和開始から再送開始まで頭打ちになっていることが読みとれる。平均要求応答時間も再送開始までクライアント台数に比例して増加していることが読みとれる。このことは、系内のトラヒック量がサーバーの処理量によって押えられていることを示している。つまり、同期式処理による通信のため、系内のトラヒック量はボトルネックとなるネットワーク構成要素の処理量によって押えられると考えられる。

クライアント台数が 50 台以降から再送が発生し始めている。しかし、この点での平均要求応答時間は 0.29[sec] であり、クライアントの再送開始の 0.7[sec] にはまだ余裕がある。これは、NFS 要求の処理時間のばらつきのため、著しく遅いと考えられるディスクアクセスを含む NFS 要求の再送が始まっていると考えられる。そして、再送が生じた台数よりクライアント台数をさらに増やすと、要求数は低下し、再送が著しく増加していく。クライアント台数 60〜70 の間に特異な現象が見られるが、この現象の解析は今後の課題とする。

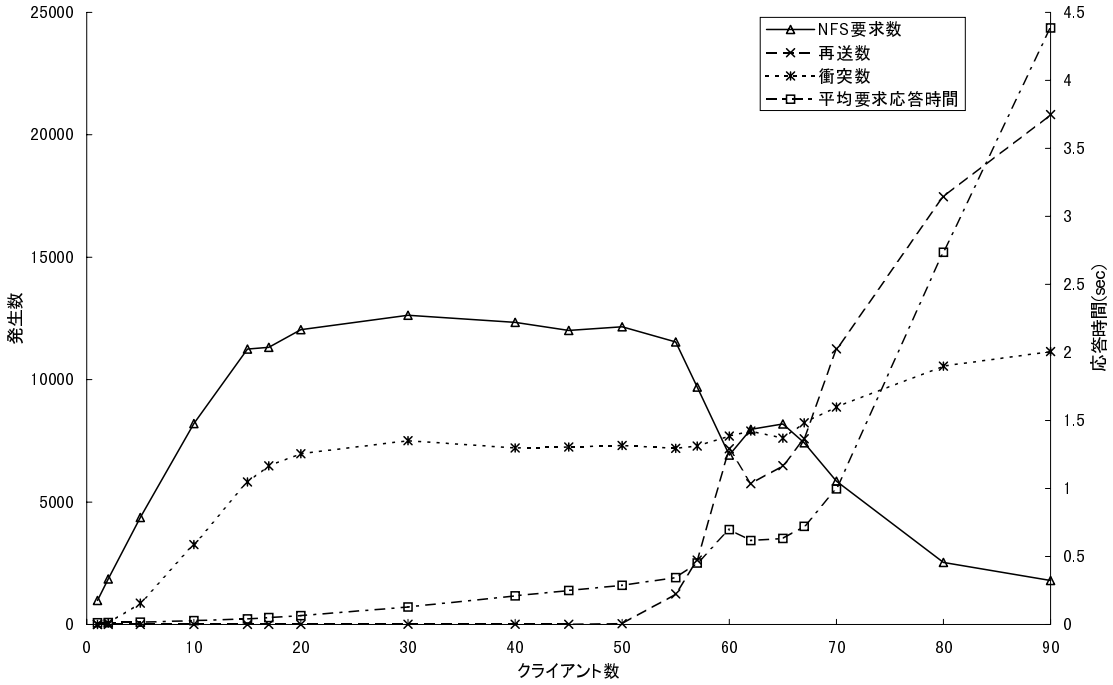

図 4.3: 系内のトラヒック量と応答時間

4.2 実験 2

実験 1 においては、システムの系の性質からサーバーボトルネックによる影響しか見ることができなかった。実験 2 は、ネットワークボトルネックを発生させるため、NFS サーバーの性能を 10 倍した時の各部の変動を解析する。

4.2.1 方法

ネットワークのトポロジーは実験 1 と同様で、単一の $10Mbit/sec$ の Ethernet セグメントに単一の NFS サーバーと複数台のクライアントが接続された場合を想定する。但し、NFS サーバーの性能を実験 1 で用いたものの 10 倍の性能のものを仮定する。変更したシミュレーターの各パラメータを以下に示す。

- サーバーモデル
  - CPU 処理時間に表 3.4 の値を$\frac{1}{10}$倍したものを用いた。
  - ディスクのアクセス時間の算出には表 3.5 のシークタイム ($a = 19.0, b = 0.197, c = 230$)、転送レート、回転待ち時間を$\frac{1}{10}$倍したものを用いた。

この状況で実験 1 と同様にクライアントの台数を変化させた場合のシミュレーションを行なった。

4.2.2 結果と考察

図 4.4 にサーバー、ディスク、ネットワークの各動作率を示す。この図から、ネットワークボトルネックとなっていることがわかる。

図 4.5 にシステム系内の NFS 要求数、再送数、そして Ethernet の衝突数と平均応答時間を示す。この図から、平均要求応答時間が非常に短いにも関わらず再送が発生していることが読みとれる。クライアント台数が 20 台、平均要求応答時間 0.006[sec] の点で再送が起こり始めている。このことは、NFS 要求応答時間に著しい変動が発生していると考えられる。そこで、NFS 要求応答時間の時間的変動を図 4.6、図 4.7、図 4.8 に示す。クライアント台数の増加によって NFS 要求応答時間の振動の振幅が増大しているのが読みとれる。

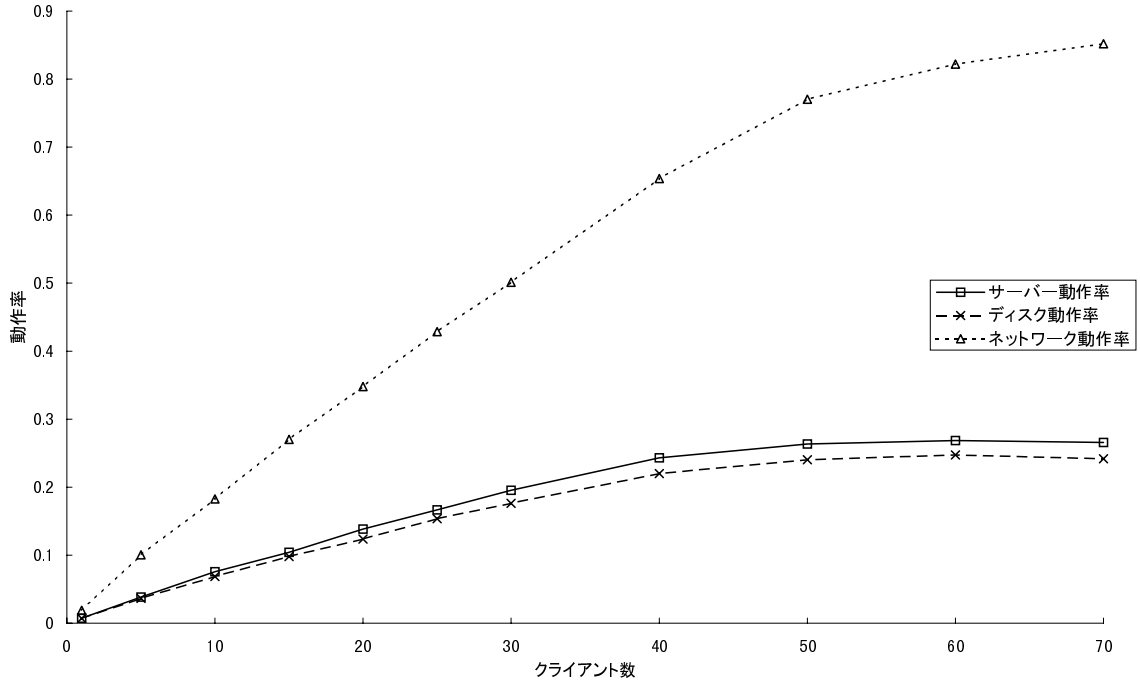

図 4.4: サーバー・ディスク・ネットワークの動作率

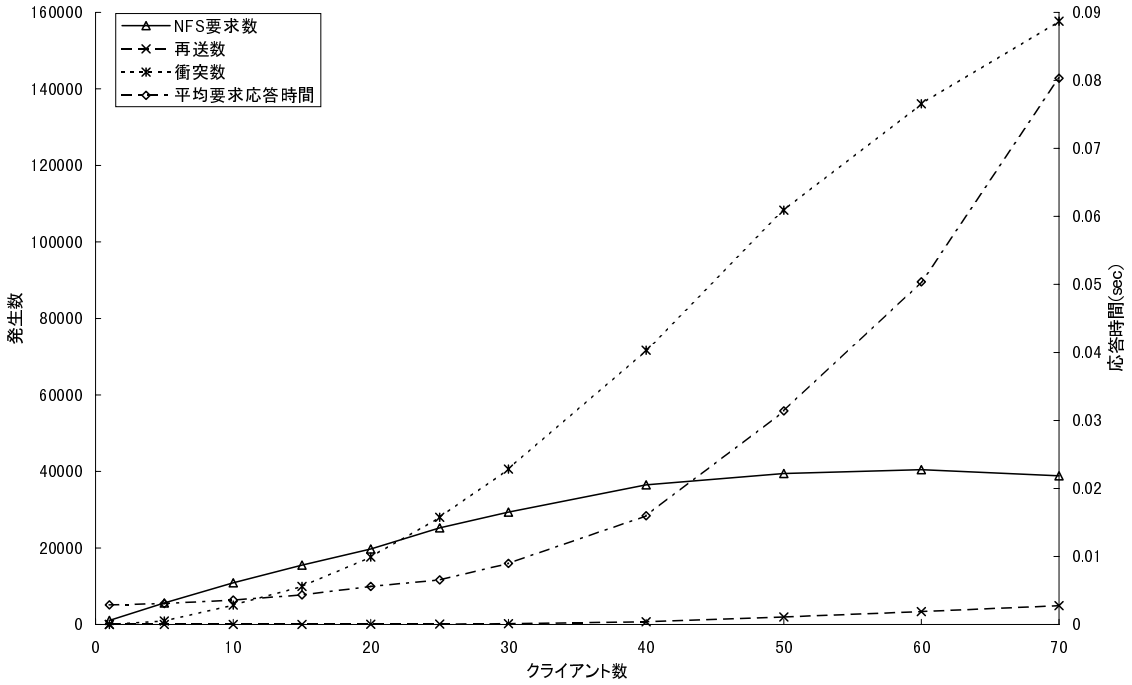

図 4.5: 系内のトラヒック変動と応答時間

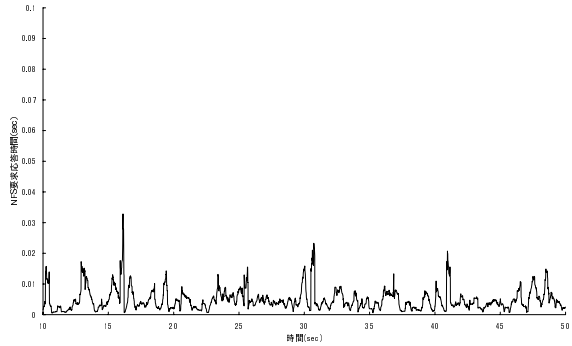

図 4.6: 20 台のクライアントにおける NFS 要求応答時間の時間的変動

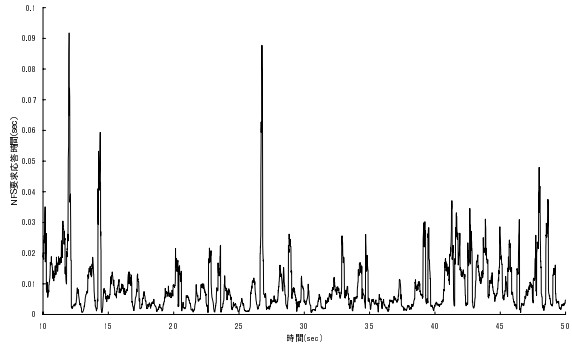

図 4.7: 30 台のクライアントにおける NFS 要求応答時間の時間的変動

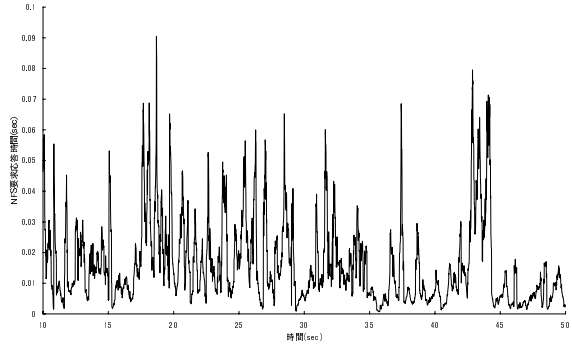

図 4.8: 40 台のクライアントにおける NFS 要求応答時間の時間的変動

このネットワーク構成における NFS 要求応答時間は、ネットワークボトルネックとなっているため、クライアントからネットワークへの出力における CSMA/CD による再送のための待ちと、サーバーの NFS 要求の処理時間、サーバーからの出力キュー待ち、サーバーからネットワークへの出力における CSMA/CD による再送のための待ちが考えられる。そして、主な応答時間の振動要因として、サーバーからネットワークへの出力の部分が考えられる。ここには、各クライアントの全ての返答メッセージによる出力キュー待ちと CSMA/CD のバックオフアルゴリズムによる再送のための待ちが発生する。

バックオフアルゴリズムについては 3.3.2 で述べた。このアルゴリズムはネットワークが混雑してくると再送のための待ち時間が大きく変動する。

図 4.9 にサーバーからネットワークへの出力における再送のための待ち時間の分布、そして同じ図の縦軸のスケールを変えたものを図 4.10 に示す。図 4.9 において、その待ち時間は 0.001[sec] 以下に集中していることを示しているが、縦軸のスケールを変えた図 4.10 から、0.007[sec] 付近にも待ち時間の集中がみられ、さらに少数ではあるが広い範囲で分布していることがわかる。この少数の通常よりも長い待ち時間が振動の原因であると考えられる。

この様子を詳しく見るために図 4.11 にサーバーがパケットを発生した時刻を横軸にとった発生状況を示す。この図において、黒い部分がサーバーがパケットを頻繁に発生した時間であり、白い部分が衝突のために比較的長く待たされた部分である。以上から、短い待ち時間による連続した送信の合間の長い待ち時間が応答時間の振動に大きく影響していると考えられる。

4.3 実験 3

通常のネットワークにおいては、ある特定のパケットのみが流れているのではなく、様々なパケットが流れていると考えられる。実験 2 において、ネットワークボトルネック時にはトラヒックの振動現象が見られることがわかったが、しかし、これまでの実験においては NFS パケットのみが流れるネットワークを仮定して行なってきた。実験 3 では、より一般化するため、他のパケットが流れているネットワークを想定して、実験 2 と同様の実験を行ない、振動現象の検証を行なう。

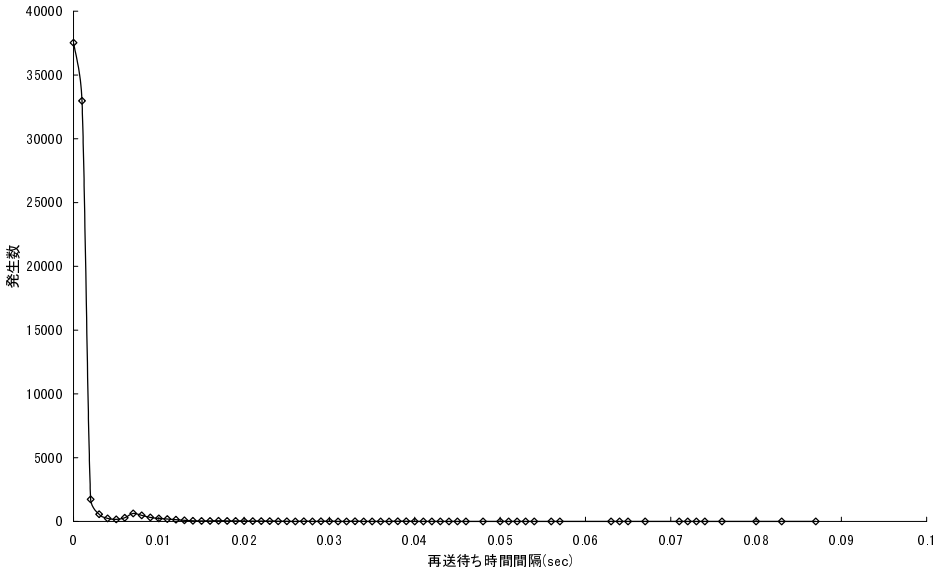

図 4.9: サーバー出力の再送待ち時間分布

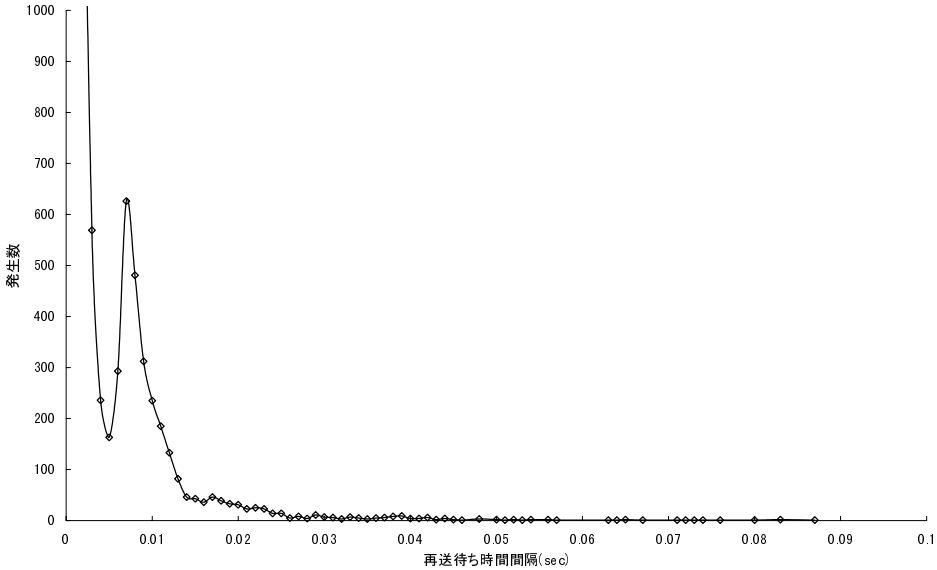

図 4.10: サーバー出力の再送待ち時間分布

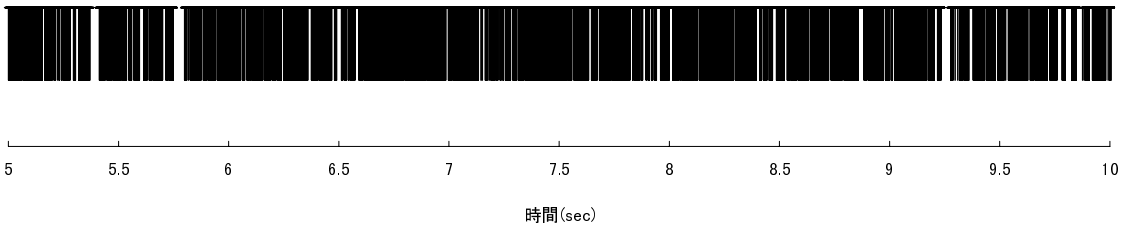

図 4.11: サーバーのパケット発生状況

4.3.1 方法

ネットワークトポロジーは実験 2 と同様で、単一の $10Mbit/sec$ の Ethernet セグメントに実験 1 の 10 倍の性能を持つ単一の NFS サーバーと複数台のクライアントが接続された場合を想定する。但し、ネットワークモデルに対して、NFS 以外のパケット（これをノイズパケットと呼ぶことにする）を出力するノードを追加する。変更したシミュレーターの各パラメータを以下に示す。

- ネットワークモデル

  NFS 以外のパケットを出力するノードの追加

  - 発生間隔は指数分布を用いてモデル化
  - パケット長は正規分布（平均：357[byte]、標準偏差：511[byte]）を用いてモデル化

この状況でノイズパケットのネットワーク占有率とクライアントの台数を変化させた場合のシミュレーションを行なった。ノイズパケットのネットワーク占有率は発生間隔に用いた指数分布の平均発生間隔を調節することによって変化させた。図 4.1 に平均発生間隔と占有率の対応表を示す。なお、パケット長を表す正規分布のパラメータは松本浩久氏によって測定されたものを用いた。

4.3.2 結果と考察

図 4.12 はノイズパケット発生率を 20%固定とし、クライアントの台数を変化させた場合の NFS 要求数、再送数、そして Ethernet の衝突数と平均応答時間を示している。実験 2 と同様に平均応答時間の非常に低い所から再送が始まっており、応答時間に振動現象が発生していると考えられる。

次にクライアント台数を固定し、ノイズパケット発生率を変化させてみた。図 4.13 にクライアント台数 20 台の時のネットワークノイズ占有率に対する NFS パケットの発生状況を表す。そして図 4.14 に発生数のスケールを変えたものを示す。この場合でも平均応答時間が短いにも関わらず再送が発生しており、応答時間に振動現象が発生していると考えられる。そこで、ノイズ占有率 20%の時の応答時間の時間的変動とノイズ占有率 40%の

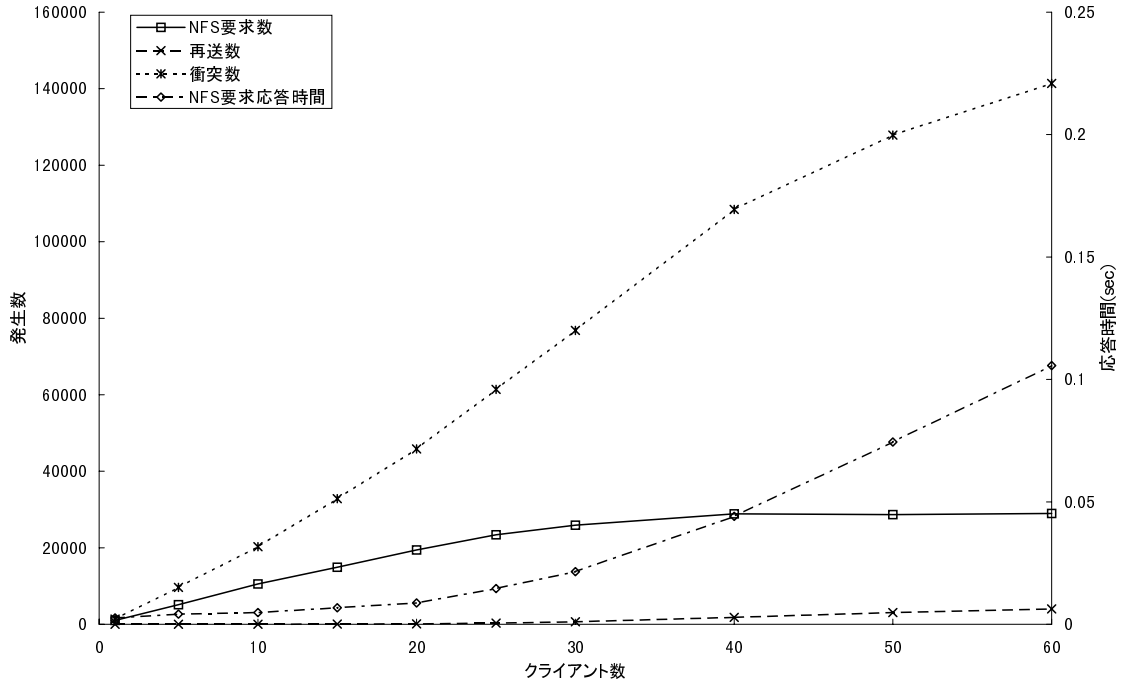

図 4.12: ノイズパケット発生率を 20%とした時の系内のトラヒック状況と応答時間

表 4.1: 指数分布における平均発生間隔とネットワーク占有率

平均発生間隔	ネットワーク占有率
4196 μsec	10 %
2098 μsec	20 %
1398 μsec	30 %
1049 μsec	40 %
839 μsec	50 %
699 μsec	60 %
599 μsec	70 %
524 μsec	80 %
466 μsec	90 %
419 μsec	100 %

時の応答時間の時間的変動を図 4.15と図 4.16にそれぞれ示す。この図からネットワークが混雑してくると同時に再送も増加していくことが読みとれる。

つまり、ネットワークボトルネックの場合、クライアント・サーバーシステムではサーバーのネットワーク出力キューに各クライアントの応答が溜ってしまう。そしてネットワークのトラヒック量の増大によって CSMA/CD の指数バックオフアルゴリズムはより長い時間領域で衝突時の待ち時間をランダムに決定する。この結果、サーバーのネットワークの出力にばらつきを与えてしまう。このばらつきによって応答時間に著しい振動現象が生じると考えられる。

4.4 まとめ

この実験によってサーバーボトルネック、ネットワークボトルネック時における各部の変動を見ることができた。

サーバーボトルネックでは、系内のトラヒック量はサーバーの処理能力によって押えられ、平均要求応答時間が再送開始時間の 0.7[sec] 付近の広い範囲まで安定したトラヒック変動であることがわかった。そして、同期式処理による通信の系内のトラヒック量は、

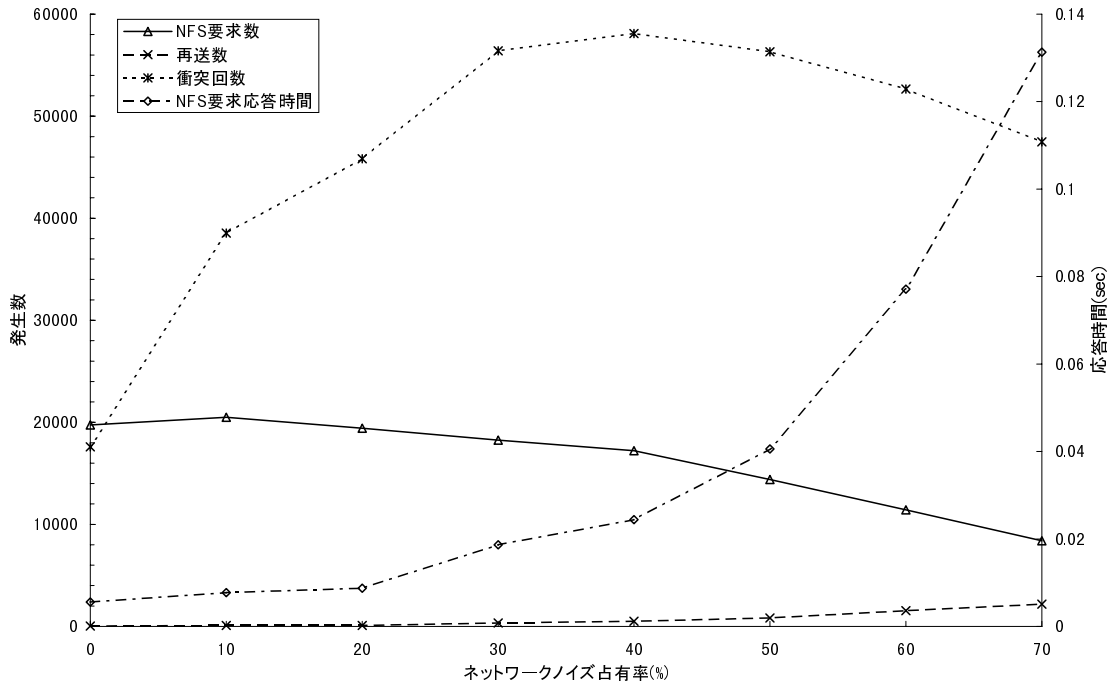

図 4.13: クライアント台数 20 台の時のネットワークノイズ占有率に対する NFS パケットの発生状況

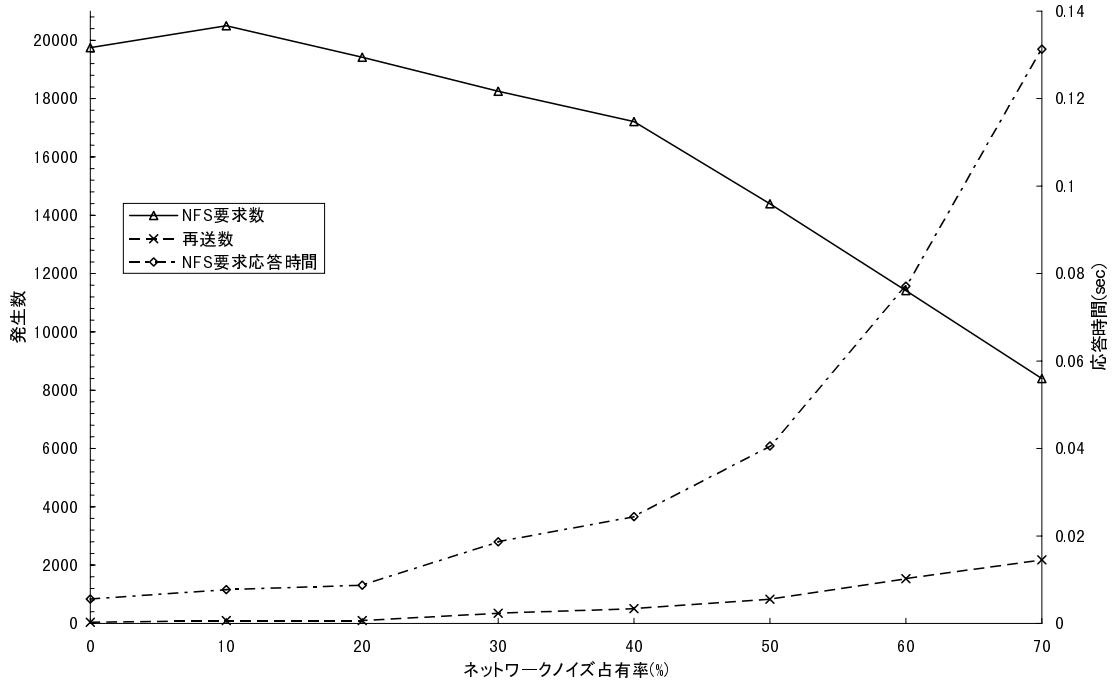

図 4.14: クライアント台数 20 台の時のネットワークノイズ占有率に対する NFS パケットの発生状況

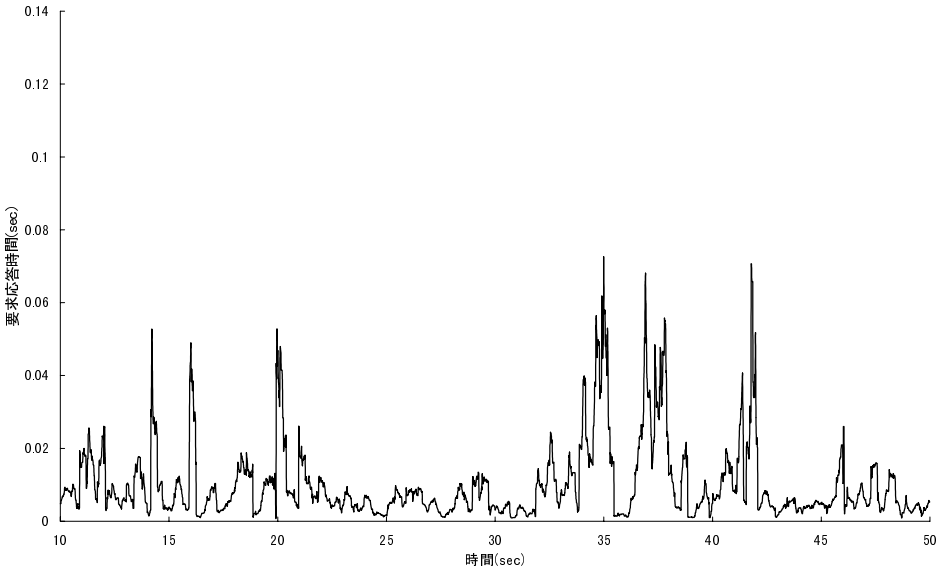

図 4.15: ノイズ占有率 20%、クライアント台数 20 台の時の応答時間の時間的変動

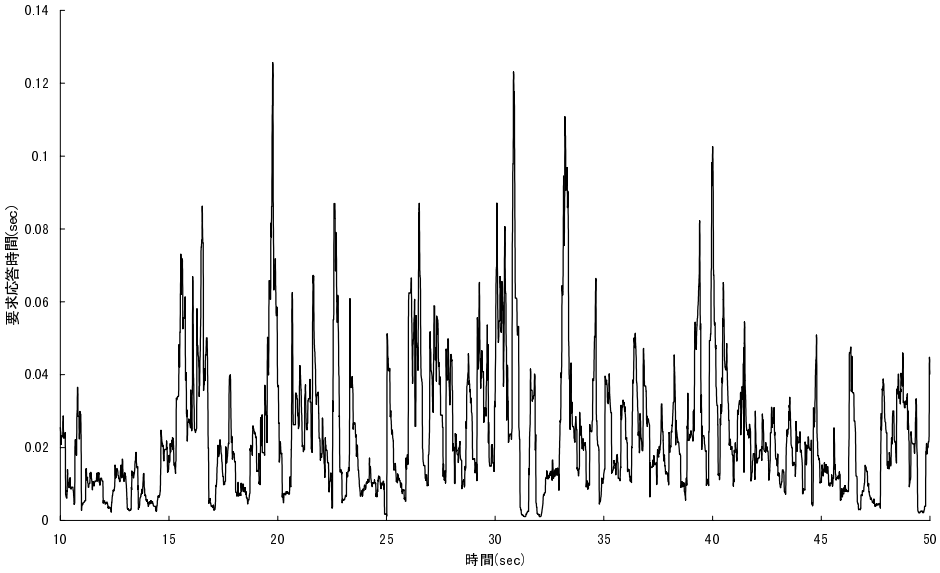

図 4.16: ノイズ占有率 20%、クライアント台数 40 台の時の応答時間の時間的変動

ボトルネックとなるネットワーク構成要素の処理量によって押えられてしまうことがわかった。

一方ネットワークボトルネックでは、ネットワークのトラヒック量によって振動現象が発生することがわかった。これは、ネットワークボトルネックのためサーバーのネットワーク出力キューに各クライアントの応答が溜っていくが、その出力は CSMA/CD による指数バックオフによってランダムに待たされてしまう。この待ち時間はネットワークが混雑すればするほど長い時間領域からランダムに決定される。この結果、出力にばらつきが生じ、応答時間に振動現象が現れてしまう。

第 5 章

まとめ

5.1 本研究の成果

クライアント・サーバーシステムによる分散ファイルシステムを対象としたシステム構成要素間の相互作用を見るために、NFS に特化したパケット発生部を有するシミュレーターを作成した。そして、サーバーボトルネック、ネットワークボトルネック時における各構成部分の変動をシミュレーションにより解析を行なった。以下に研究の成果を示す。

シミュレーターの作成

1. NFS に特化したトラヒックモデルのモデル化を行なった。トラヒックの発生をユーザーの思考間隔時間とアプリケーションによる NFS 要求群の 2 つに分割して考え、ユーザーの思考間隔時間を正規分布、アプリケーションによる NFS 要求群の発生順序を確率状態遷移行列を用いて、乱数により発生するトラヒックモデルを作成した。

2. シミュレーターモデルの各パラメーターを実測値により決定した。

3. 実際のプロトコルの動作と同様に動作するようにモデル化を行なった。

シミュレーション実験

1. サーバーボトルネック、ネットワークボトルネック時における各構成部分の変動を分析した。

2. サーバーボトルネックでは、RPC の同期式の処理によるメッセージ通信によってトラヒック量がサーバーの処理性能に押えられることがわかった。これによって、トラヒック量はボトルネックとなるネットワーク構成要素の処理性のに依存することがわかった。

3. ネットワークボトルネックでは、クライアント・サーバーシステムと CSMA/CD によるバックオフによって応答時間に激しい振動を与えることがわかった。

5.2 今後の課題

今後の課題を以下に示す。

1. 実験 1 において、再送が発生し始めてからクライアント台数をさらに増やすと要求数は低下し、再送数は著しく増加していく傾向を示していた。しかし、クライアント台数 60〜70 の間で特異な現象が見られていた。これは、再送を発生させるタイムアウトが 0.7 秒と固定値であるため、応答の長いディスクアクセスを伴う要求と、応答の短いディスクアクセスを伴わない要求の発生比率が変動していると考えられるが、原因についてはさらに十分な解析が必要である。

2. シミュレーターのモデル化において、NFS サーバーを単一プロセッサ、単一ディスク装置とした。しかし、一般的な NFS ファイルサーバーは処理の高速化・大容量化のため、複数プロセッサ、複数ディスク装置となっているのが普通である。このようなアーキテクチャによるモデル化を考慮する必要がある。

謝辞

本研究を進めるにあたり大変多くの人々に助けられました。

日頃からあたたかく御指導下さいました指導教官日比野靖教授に心から感謝致します。また、有益な御指導、御指示を下さいました堀口進教授、横田治夫助教授に深く感謝致します。

丹康雄博士には日頃から有益な御指導や御助言をいただき大変ありがとうございました。また、トラヒック測定において、測定方法や測定装置についての御指導、御協力をいただき深く感謝致します。

杉野栄二博士には日頃から有益な御助言を頂き、また、多方面に渡って励まして頂き大変ありがとうございました。

宮崎純博士、味松康行氏、小林浩之氏にはディスク装置のモデル化においての有益な御助言をいただき深く感謝致します。

松本浩久氏には同じ研究分野であることから本研究全般にわたっての討論を行うことができ、大変有益な助言、指摘をいただくことができました。深く感謝致します。

小川知之氏、小林純也、杉本朗子氏、西田悦雄氏、松田理絵子氏、研究室の諸兄には日頃から有益な助言をいだだきありがとうございました。

最後に、ここまで暖かく見守ってくれた家族に感謝致します。

参考文献

[1] 石原 進、岡田 稔, “負荷集中 LAN におけるクライアントサーバシステムの評価”, 信学技報, IN96-129, 1997

[2] 石田 陽子、高原 幹夫, “イーサネット上のトラヒック特性によるパケット発生モデルについて”, 電子情報通信学会論文誌, Vol.78-B-I,No.11,pp.664-671,1995

[3] W.Richard Stevens 著、橘 康雄 訳, “詳解 TCP/IP (TCP/IP Illustrated,Volume 1:The Protocols)”, ソフトバンク, 1997

[4] 秋丸春夫、Robert B.Cooper, “通信トラヒック工学”, オーム社, 1985

[5] M.Santifaller 著、野間 泉、高橋伸彰、西田佳史、藤原一博 訳, “TCP/IP と NFS：UNIX でのインターネットワーキング (TCP/IP AND NFS:Intern etworking in a UNIX Environment)”, トッパン, 1993

[6] Carl Malamud 著、相田 仁、小池汎平 共訳, “SUN ネットワーク詳説”, 丸善, 1993

[7] “データ通信”, 電子通信学会, 1986

[8] Edward K Lee、Randy H. Katz, “The Performance of Parity Placements in Disk Arrays”, IEEE TRANSACTIONS ON COMPUTERS, VOL. 42, NO. 6, JUNE 1993

[9] 桐山 光弘, “「待ち行列」がわかる本”, 日刊工業新聞社1997

[10] Hal Stern 著、倉骨 彰 訳、砂原 秀樹 監訳, “Managing NFS and NIS”, アスキー出版局1992

[11] 石川 宏, “C によるシミュレーションプログラミング”, ソフトバンク1994