2004 年度

オープンソースソフトウェア活用基盤整備事業

「OSS 性能・信頼性評価 / 障害解析ツール開発」

LKST を利用した
カーネル性能評価ツールの考察

独立行政法人 情報処理推進機構

商標表記

- Linux は、Linus Torvalds の米国およびその他の国における登録商標あるいは商標です。
- MIRACLE LINUX は、ミラクル・リナックス株式会社が使用許諾を受けている登録商標です。
- その他記載の会社名、製品名は、それぞれの会社の商号、商標もしくは登録商標です。

目次

1. 開発の背景

1.1. 開発動機

サーバ分野において、Linux を適用したシステムが普及・拡大している。数年前までは、Web サーバやメールサーバ、ネームサーバといったネットワークのフロントエンドサーバへの適用が中心であったが、最近では、アプリケーションサーバ、DB サーバから構成されるエンタープライズシステムへの適用ニーズも出てきている。

エンタープライズシステムでは障害や性能遅延が発生した場合、迅速な対応を求められるが、Linux にはダンプやトレースといった解析のための標準的なツールが無く、各社固有のノウハウで対応しているのが現状である。

上記のうちの障害解析に関しては、カーネル内部で起きているイベントを記録する機能として LKST（Linux Kernel State Tracer）(*)の開発を進めている。この LKST によりパニックなどの障害に関する解析を支援できるようになったが、性能評価・解析に関しては適切なツールがなく、ベンチマークでの評価に頼る形になっている。この方法では具体的にどの部分のプログラムやアルゴリズムに問題があるかが特定しにくく、異なるバージョンのカーネルで性能を比較しながら試行錯誤するしか方法がない。性能障害発生時に十分なデータが得られないため、解決に時間がかかり、原因の特定に至らないケースも多々ある。このため、もっと精密にかつ容易に性能評価を行なう機能が望まれている。

そこで、LKST の情報収集機能を利用し、性能の評価を行なう機能（ツール）を開発することとした。

*)日立、IBM、富士通、NEC の 4 社協業「Enterprise Linux の信頼性の強化の推進」の一環として開発プロジェクトを開始。開発したソースコードやドキュメントは SourceForge（http://lkst.sourceforge.net, http://lkst.sourceforge.jp） にて公開している。

1.2. 従来の LKST の特徴

LKST は、クラッシュするような障害を解析するために、カーネル内の処理の遷移情報を得ることを目的に開発が進められており、カーネル内部で発生したイベントの情報を発生順に記録し保持する機能を有している。記録したイベントに関しては、解析者が自由に加工しやすいように、ログ内容をテキスト形式に変換する機能を提供している。LKST はカーネル内情報を記録するための、柔軟で拡張性のある、以下の特徴を持つ。

(1) 取得するイベントの種類を容易に追加可能な構造
(2) 取得するイベントを動的に選択する機能

(3) イベント取得時に、記録する情報を加工する機能
(4) 情報を記録するバッファのサイズを動的に変更する機能
(5) 例外発生時でもイベントの記録が可能な構造

1.3. 従来の LKST の課題

Linux カーネルの性能を評価するためには、Linux カーネル内の制御の要となる部分の性能情報が必要である。性能評価に有用な情報を表 1.3-1 に示す。これらの情報があれば、大抵の性能を評価でき、性能障害時には原因究明のヒントを得ることができる。

しかし、従来の LKST には、表 1.3-1 の情報を収集・評価する手段がない。そこで、その手段を実現することを今回の課題とする。

表 1.3-1 性能評価に有用な情報

項番	収集情報区分	内容
1	システムコール関係	アプリケーションからの要求に対する性能情報
2	割り込み関係	ソフトウェア割り込み要求処理に対する性能情報
3	メモリ関係	カーネル内でのメモリ要求・解放要求処理に対する性能情報
4	ファイル IO 関係	ブロック IO 処理に対する性能情報
5	ネットワーク IO 関係	パケット送信要求処理に対する性能情報
6	プロセススケジュール関係	ユーザ／カーネルプロセスのスケジュール処理に対する性能
7	ロック関係	ロックの発生頻度と待ち時間

1.4. 本ツールの想定利用者について

本ツールの使用に当たっては、OS についての基本的な知識を持っていることを前提としている。本ツールの利用者としては、カーネル開発者、システムエンジニア、ミドルウェア開発者などを想定している。

カーネル開発者：本ツールを使うことで、現行カーネルのボトルネック解析、新しいアルゴリズムや実装のミクロな評価などを行うことが出来る。

システムエンジニア：本ツールを使うことで、システムのスローダウンの原因が OS にある場合にその根源を探るために利用することが出来る。

アプリケーション開発者：本ツールを使うことで、アプリケーションを Linux 上で動かした場合のシステムコールから先の振る舞いを検証することが出来る。

2. 開発した機能

2.1. LKST 性能評価ツール (lkstlogtools)

2.1.1. 性能評価に関する要求

図 2.1-1 に示すように、LKST はすでにカーネル情報を収集するための仕組みを備えている。そこで、その仕組みを利用して性能評価のための情報収集を行ない、解析するツールの開発が必要である。具体的には、従来の LKST で表 1.3-1 の情報を取得するために、図 2.1-1 に示した 2 項目を実現する必要がある。

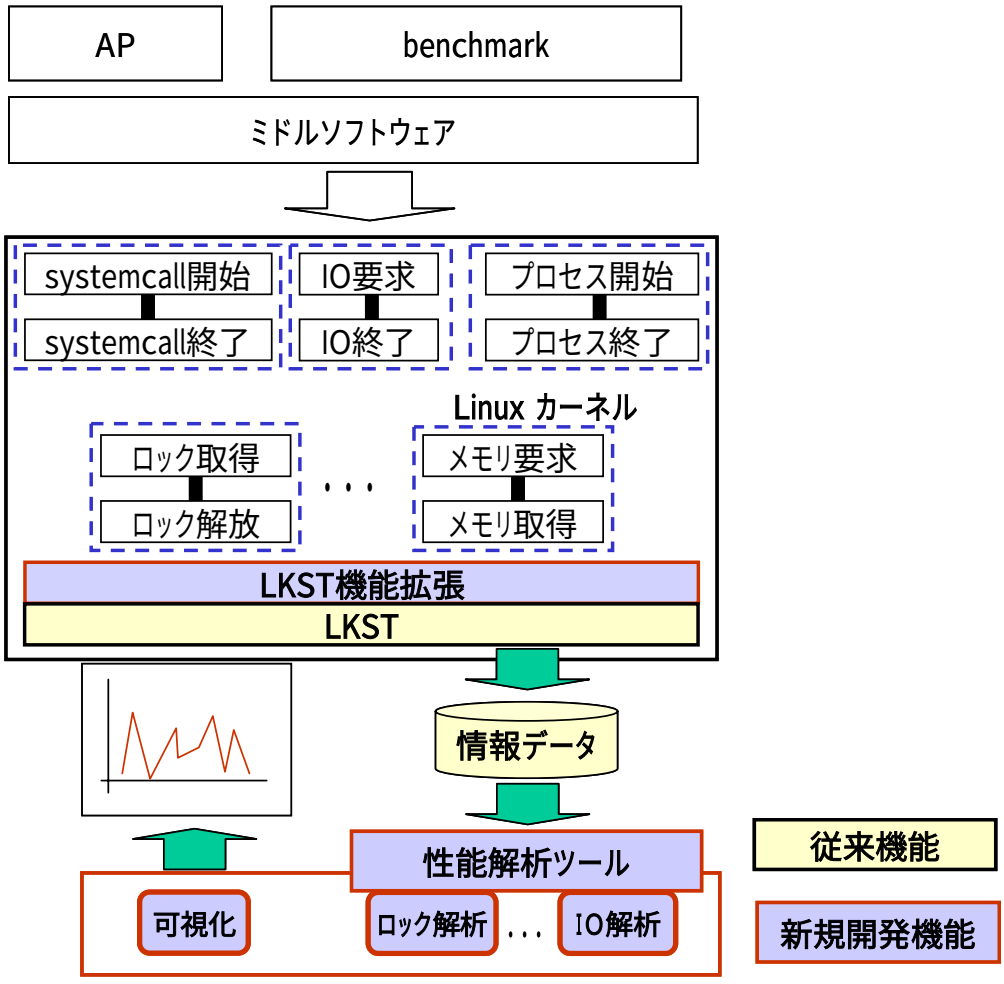

図 2.1-1 LKST による性能評価機能の構成

2.1.1.1. LKST の機能拡張

(1) 性能測定に特化したイベント種類の追加
(2) 性能測定に特化したイベント加工機能の追加

2.1.1.2. LKST 性能解析ツール

(1) 取得したイベントのうち、性能に関する情報だけを取り出す機能
(2) 取得したイベントのうち、対になるイベント同士を結びつけ、イベント間の時間などを取り出す機能
(3) 上記の情報を、時系列情報、統計情報、分布情報などにまとめて、解析を補助する機能
(4) 上記の情報を、グラフとして可視化し、解析を補助する機能

2.1.2. LKST の機能拡張の開発

「2.1.1.1 LKST の機能拡張」要求に対し、カーネルから性能評価情報を取得する機能として、LKST に対して以下のような拡張機能を開発する。

(1) 性能測定用の機能を LKST に追加するためのパッチの開発
  (a) LKST を適用したカーネルに適用するパッチ群（hook-*.patch）
  (b) LKST 操作用ツールに適用するパッチ（lkstutils-logformat.patch）

上記の(a)は Linux カーネルに対するフックポイント（測定箇所）の追加であるが、この拡張開発は、従来の LKST が提供している“ フックポイントの追加方法 ”を活用して行なった。“ フックポイントの追加方法 ”については、SourceForge の LKST ホームページで公開している howto.txt に記載されている。

また(b)は、(a)で追加した TSC(Time Stamp Counter)情報と、その TSC をシステム時間に変換するための情報を扱う処理の追加であり、情報形式を解釈して処理している lkstbuf と lkstlogd を拡張開発した。

なお、測定項目と箇所に関しては、表 1.3-1 の内容に一致した部分をカーネルソースコードから探して決定した。その内容を表 2.1-1 に示す。

表 2.1-1　測定項目と箇所

項番	収集情報区分	測定項目	測定箇所
1	システムコール	処理時間	システムコールの入口と出口
2		使用傾向	システムコールの入口
3	ソフトウェア割り込み	実行遅延時間	ソフトウェア割り込み要求の実行場所
4		応答性能	タイマ割り込みの実行場所
5	メモリ確保	処理時間	メモリ確保の入口と出口
6	メモリ解放	処理時間	メモリ解放の入口と出口
7	ブロック IO	処理時間	ブロック IO の入口と出口
8		IO リクエストキュー	ブロック IO の入口
9	ネットワーク IO	処理時間	ネットワーク IO の入口と出口
10		通信サイズ傾向	ネットワーク IO の入口
11	ロック	競合傾向	ロック取得待ちの開始と終了
12		ロック時間	ロック取得とロック解放
13	スケジューラ	ランキューの傾向	スケジューラの入口
14		ウェイトキューの傾向	ウェイトキューの入口と出口
15		ウェイト処理遅延時間	ウェイトキューの入口と出口
16		スケジューラ処理時間	スケジューラの入口と出口
17	プロセス	プロセスの生存時間	フォーク、プロセス終了
18		プロセスの状態遷移	フォーク、コンテキストスイッチ、ウェイクアップ、プロセス終了

(2) 性能測定用にイベント内容を加工し記録する拡張イベントハンドラの開発
  (a) ウェイトキューの情報を取得するハンドラ（lksteh_wqcounter.ko）
  (b) プロセスの状態遷移に関する PID を取得するハンドラ（lksteh_procstat.ko）
  (c) システムのメモリ管理情報を取得するハンドラ（lksteh_sysinfo.ko）

上記の拡張開発は、従来の LKST が提供している“イベントハンドラの作成方法”を活用して行なった。“イベントハンドラの作成方法”については、SourceForge の LKST ホームページで公開している howto.txt に記載されている。

2.1.3. LKST 性能解析ツールの開発

「2.1.1.2 LKST 性能解析ツール」要求に対し、「2.1.2 LKST の拡張機能の開発」で開発した機能を使って収集した情報を解析するため、以下の機能を持つ解析ツール群を新規に開発した。

(1) イベント解析ツール

LKST で取得したイベントを解析するツールを表 2.1-2 に示す。

表 2.1-2　イベント解析ツール一覧

コマンド名	機能説明
lkstla	性能測定用のイベントから、性能情報（キューの長さなど）を取り出し、対になるイベント(システムコールの始まりと終わりなど)同士を結びつけ、イベント間の時間などを取り出す。 またこれらの情報を、時系列情報、統計情報、分布情報として表示する。
lkstlogdiv	LKST の収集データのファイル内において、時間の整合性を取り、ファイルを分割する。
lkst_fmt_sysinfo	LKST が収集するシステム情報（メモリ使用率など）を解析する。

(2) 可視化ツール

解析ツールにより得られた情報をグラフとして可視化するツールを表 2.1-3 に示す。

表 2.1-3　可視化ツール一覧

コマンド名	機能説明
lkst_plot_log	lkstla で解析した時系列ログの可視化
lkst_plot_stat	lkstla で解析した統計結果の可視化
lkst_plot_dist	lkstla で解析した対数分布結果の可視化
lkst_plot_sysinfo	lkst_fmt_sysinfo で解析したシステム情報の時系列ログの可視化

(1)(2)のツールの詳細に関しては SourceForge にある LKST の開発サイトにおいて公開している、性能評価ツール（lkstlogtools）のマニュル（使用説明書）に記載している。

2.2. LKST 性能評価機能の MIRACLE LINUX V3.0 への移植

LKST 性能評価機能の kernel 2.4 対応を行うにあたり、MIRACLE LINUX V3.0 への移植を実施した。

2.2.1. 作業項目

MIRACLE LINUX V3.0 には既に LKST v2.0.1 が組み込まれている。従って、LKST 性能評価機能の kernel 2.4 対応として次の開発作業を実施した。

(1) LKST の最新バージョンである v2.2.1 へのアップデート
(2) kernel へ性能評価機能用の新規フックポイントの追加、情報収集用のカーネルモジュール追加
(3) LKST 解析用ツール(lkstutils)への性能評価機能の追加

2.2.2. 作業成果

LKST 性能評価機能の kernel 2.4 対応のための移植作業の結果、次の成果が得られた。

(1) MIRACLE LINUX V3.0(kernel 2.4)で LKST 性能評価機能を利用することが可能になった。
(2) LKST 性能評価機能を組み込んだ kernel、および lkstutils を RPM パッケージとして作成した結果、システム管理者にとって日常的なシステム管理作業であるパッケージのアップデート作業のみで、LKST 性能評価機能を MIRACLE LINUX V3.0 に導入できるようになった。
(3) LKST 性能評価機能を、サーバ用 OS として普及している最新版の MIRACLE LINUX で容易に導入可能になったことにより、LKST 性能評価機能の普及に貢献できるようになった。

2.2.3. Kernel 2.4 への移植のポイント

今回の開発作業において遭遇した技術的なポイントを説明する。

2.2.3.1. kernel の開発

LKST 本体の開発は既に kernel 2.6 ベースになっており、kernel 2.4 ベースの MIRACLE LINUX V3.0 に適用するためには、LKST 本体のパッチを修正する必要がある。LKST 本体のパッチの kernel 2.4 への対応は、MIRACLE LINUX V3.0 の開発時点で LKST v2.0.1 の導入を行い完了していたが、性能評価機能に対応する LKST として最新バージョンの v2.2.1 を利用するため、今回の作業において、LKST v2.2.1 にアップデートを行った。

さらに今回開発した性能評価機能として、カーネル内部のイベントを記録するためのフックポイントの追加作業を行った。

これらの移植作業の過程において、次のような技術的なポイントがあった。

(1) LKST v2.0.1 から v2.2.1 にアップデートするにあたり、LKST v2.2.1 で kernel 2.6 の機能を利用している部分を kernel 2.4 に対応するために、LKST のコア部分の修正が必要であった。
特にデバイスファイルの登録処理に関して、大きく変更する必要があったが、LKST v2.0.1(MIRACLE LINUX V3.0 版)など、kernel 2.4 に対応した以前のバージョンの LKST の実装を参考にすることで、移植の難易度を下げることができた。

(2) システムコールのエントリポイントとして、kernel 2.4 では SYSCALL_ENTRY/SYSCALL_EXIT のみであるので、kernel 2.6 で導入された SYSCALL_SYSENTER/SYSCALL_SYSEXIT に関連する処理を削除する必要があった。

(3) CPU 数に関連する実装が変更されているため、LKST で CPU 数に関連する実装部分を kernel 2.4 用に変更する必要があった。

(4) ほとんどのフックポイントに関しては、kernel 2.6 のコードと同じ箇所に追加することで対応できたが、メモリ関連、ブロック I/O 関連のフックポイントについては、実装が大きく異なるため、kernel 2.6 のコードを理解したうえで、kernel 2.4 へのフックポイントの追加場所を定める必要があった。

(5) タイマ関連の情報を取得するために追加したフックポイントで、時刻情報取得の実装が kernel 2.4 と kernel 2.6 で異なっており、カーネルの互換性を保ったまま LKST の実装にあわせるために do_gettimeoffset()の修正が必要であった。

(6) LKST のフックポイントには 2 種類あり、NORMAL 形式と、INLINE 形式のどちらかを選択する必要がある。メモリ管理ルーチンなどにおいて、kernel 2.6 では処理が 1 つのサブルーチン内で行われているものが、kernel 2.4 では似たような処理を複数のサブルーチンで行っているために、フックポイントの形式を NORMAL 形式から INLINE 形式に変更して、複数の箇所に同じフックポイントを追加する必要のある箇所があった。

2.2.3.2. lkstutils の機能強化

今回開発した性能評価機能を利用するために、解析用ツールの lkstla などいくつかのコマンドやスクリプトを新たにユーザに提供する必要があった。

現在、MIRACLE LINUX V3.0 では、LKST 用のツールとして lkstutils パッケージを提供しており、今回開発した機能も同じパッケージ内に同梱することにした。これは、ユーザの利便性を考慮して、今後の kernel パッケージのアップデートおよび、lkstutils パッケージのアップデートで、今回開発した LKST 性能評価機能を簡単に導入可能にしたいと考えたからである。

lkstutils への性能評価機能の追加作業の過程においては、次のような技術的なポイントがあった。

(1) 従来の lkstutils パッケージは、lkst のソースコードの一部分である lkstutils のみから作成していたが、今回の性能評価機能は lkst 全体のソースコードをもとにビルドする仕組みとなったため、lkstutils パッケージ自体を lkst 全体のソースコードをもとにビルドする仕組みに変更した。

(2) シンボル情報の取得先を kernel 2.6 の/proc/kallsyms から、kernel 2.4 の/proc/ksyms に変更した。これにともない、取得できるシンボル情報の精度が kernel 2.6 と比較すると低下した。例えば、モジュール内部の static 関数などは、モジュール名とアドレスのみから表示されるようになり、利便性が多少低下した。

(3) LKST のソースコードでは、make および make install でツール、カーネルモジュールをビルド・インストールする仕様となっているが、カーネルモジュールは kernel パッケージにパッチとして適用済みのため、ツールのみをビルドするように make 時のオプションを変更した。

3. 開発した機能の有効性の検証と考察

3.1. LKST 性能評価機能の各機能の有効性の評価

LKST 性能評価機能の各機能の有効性を評価するために、LKST 性能評価機能を利用してカーネル内部の詳細なデータを取得した。データの取得は、カーネルビルド、および chat ベンチ実行中に行った。

カーネルビルドには、オリジナルの kernel 2.4.28 を利用した。
http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.28.tar.bz2

chat ベンチは、次の URL からダウンロード可能である。
http://lbs.sourceforge.net/chat/chat-1.0.1.tar.gz

3.1.1. 評価環境

(1) システム環境

評価を行ったシステム環境を表 3.1-1 に示す。

表 3.1-1 評価を行うシステム環境

項番	項目		評価環境
1	ハードウェア	CPU	Xeon 2.80GHz HT × 1 個
2		メモリ	1GByte
3		ハードディスク	36GB(U320 SCSI)
4	ソフトウェア	カーネル	MIRACLE LINUX V3.0 +新 LKST
5		解析ツール	lkstla
6		負荷ツール	カーネルビルド、chat ベンチ

(2) LKST 実行条件

評価を行う際の、LKST の実行条件は以下の通り。

(a) 採取するイベントは LKST 性能評価機能で利用されるイベント
(b) 使用するハンドラは通常ハンドラと、lkst_wqcounter
(c) イベントの記録には lkstlogd を使用

3.1.2. システムコールの呼び出し

カーネルビルド時のシステムコールの呼び出し状況を、LKST 性能評価機能を利用して、データを取得した。

LKST 性能評価機能を利用して得られた各システムコールの呼び出し状況の統計情報を次のコマンドで取得した。

```
# lkstla syscall -s sebuf00.0
```

取得した結果を、表 3.1-2 に示す。

表 3.1-2 カーネルビルド時のシステムコールの呼び出し状況の統計情報

sysno	syscall_name	count	average	max	min
3	read	8034	0.023076387	90.083256507	0.000000367
4	write	3152	0.000220853	0.081517976	0.000000417
5	open	16032	0.000045013	0.202784281	0.000001509
6	close	14662	0.000001250	0.000027938	0.000000187
7	waitpid	1658	0.031027354	12.099844293	0.000000769
11	execve	1657	0.000292946	0.153153177	0.000001854
12	chdir	1786	0.000007886	0.009594341	0.000001326
13	time	385	0.000001162	0.000004169	0.000000350
30	utime	166	0.000023495	0.000044448	0.000017284
33	access	21878	0.000006903	0.020774517	0.000001193
36	sync	2	0.150531058	0.301052424	0.000009692
42	pipe	663	0.000010388	0.000028874	0.000003553
45	brk	3830	0.000000742	0.000021612	0.000000180
54	ioctl	398	0.000002674	0.000122255	0.000000458
63	dup2	2025	0.000001116	0.000007300	0.000000331
90	mmap	8164	0.000003235	0.000034933	0.000001132
91	munmap	5181	0.000006453	0.000075592	0.000002781
114	wait4	216	0.471461605	13.441783911	0.000004100
118	fsync	4	0.008649823	0.022778521	0.000000621
120	clone	888	0.000226905	0.000490256	0.000142467
122	uname	1243	0.000001121	0.000020207	0.000000337
142	_newselect	586	0.504029704	29.999847702	0.000000805
162	nanosleep	3	0.028387313	0.029999399	0.025168611
174	rt_sigaction	13566	0.000000647	0.000019189	0.000000132
175	rt_sigprocmask	17994	0.000000497	0.000018929	0.000000252
180	pread64	61	0.000171169	0.008869589	0.000001675

183	getcwd	548	0.000002880	0.000014088	0.000001823
190	vfork	195	0.191333717	2.928745128	0.000280941
192	mmap2	6014	0.000067530	0.365378266	0.000001200
195	stat64	11837	0.000046106	0.040002325	0.000000892
196	lstat64	4911	0.000237922	0.016212992	0.000000777
197	fstat64	10108	0.000000566	0.000004156	0.000000383
200	getgid32	346	0.000000140	0.000000813	0.000000119
220	getdents64	2836	0.001566475	1.196743794	0.000000609
221	fcntl64	4605	0.000000578	0.000005340	0.000000315
243	set_thread_area	897	0.000001011	0.000004758	0.000000518

この表の項目は、左の列から順に次のデータを表している。

(1)システムコール番号
(2)システムコール名
(3)記録された回数
(4)平均実行時間
(5)最大実行時間
(6)最小実行時間

表 3.1-2 で取得したデータを、lkst_plot_stat コマンドを実行してグラフに変換したものを図 3.1-1 に示す。

```
# lkst_plot_stat syscall.stat
```

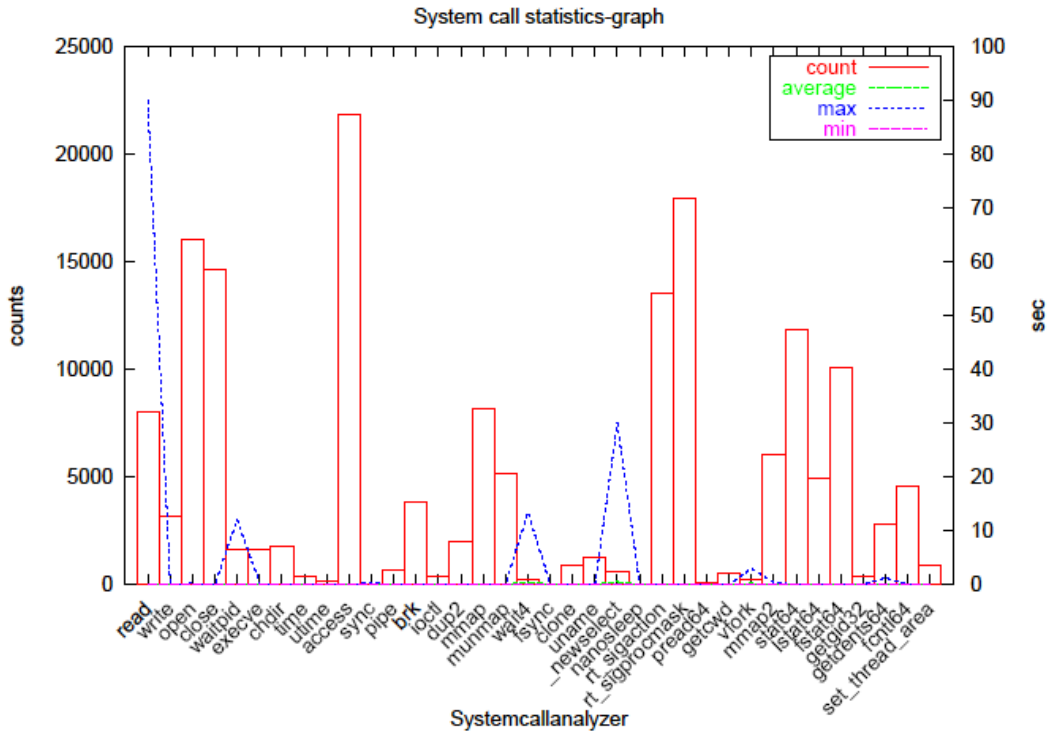

図 3.1-1 システムコールの呼び出し状況の統計グラフ

図 3.1-1 は、横軸がシステムコールの種類、棒グラフで表される縦軸がそのシステムコールが記録された回数、折れ線で表される縦軸がそれぞれのシステムコールの平均実行時間、最大実行時間、最小実行時間を表す。

この統計データをもとに、呼び出し回数(counts)、平均実行時間(average)より、カーネルビルド時にプロセスの elapse time を占める割合が多いシステムコールを抜き出した結果を、 表 3.1-3 に示す。

表 3.1-3 カーネルビルド時にプロセスの elapse time に占める割合の高いシステムコール

sysno	syscall_name	count	average	max	min
142	_newselect	586	0.504029704	29.999847702	0.000000805
3	read	8034	0.023076387	90.083256507	0.000000367
114	wait4	216	0.471461605	13.441783911	0.000004100
7	waitpid	1658	0.031027354	12.099844293	0.000000769
190	vfork	195	0.191333717	2.928745128	0.000280941
220	getdents64	2836	0.001566475	1.196743794	0.000000609
196	lstat64	4911	0.000237922	0.016212992	0.000000777
5	open	16032	0.000045013	0.202784281	0.000001509

4	write	3152	0.000220853	0.081517976	0.000000417
195	stat64	11837	0.000046106	0.040002325	0.000000892

システムコールの呼び出し状況のデータにはカーネル内での待ち時間も含まれるため、select(2)や wait(2)などは回数が少ないが、平均実行時間が長いために上位に現れている。一方、open(2)や stat(2)などは、1 回あたりの実行時間は短いが、非常に頻繁に呼び出されるために、全体の実行時間に与える影響が大きいことが分かる。

このように、LKST 性能評価機能を利用することで、カーネルのシステムコール実行時間に関する情報を取得できることを確認できた。

3.1.3. ソフトウェア割り込み

カーネルビルド時のソフトウェア割り込みの遅延状況を、LKST 性能評価機能を利用して計測し、次のコマンドでデータを取得した。

```
# lkstla softirq -s sebuf00.0 > softirq.stat
```

取得したデータを表 3.1-4 に示す。

表 3.1-4 ソフトウェア割り込みの遅延状況の統計

irqno	softirq_name	count	average	max	min
0	tasklet hi	11871	0.000000547	0.00004786	0.000000375
2	net_tx	843	0.000003282	0.00001507	0.000002956
3	net_rx	9557	0.000003289	0.000161771	0.000002852
4	scsi	6	0.000000668	0.000000821	0.000000611

表 3.1-4 の各項目は、それぞれ左より次の意味を表している。

(1)ソフトウェア割り込みの登録番号
(2)ソフトウェア割り込みの種類
(3)記録された回数
(4)ソフトウェア割り込みの割り込み発生から、実際に実行が行われるまでの遅延時間の平均
(5)ソフトウェア割り込みの割り込み発生から、実際に実行が行われるまでの遅延時間の最大
(6)ソフトウェア割り込みの割り込み発生から、実際に実行が行われるまでの遅延時間の最小

測定結果のデータより、tasklet_hi(タスクレット)の遅延時間の対数分布結果を、lkst_plot_dist を利用してグラフ化したものを図 3.1-2 に示す。

```
# lkstla softirq -d sebuf00.0 > softirq.dist
# lkst_plot_dist softirq.dist tasklet_hi
```

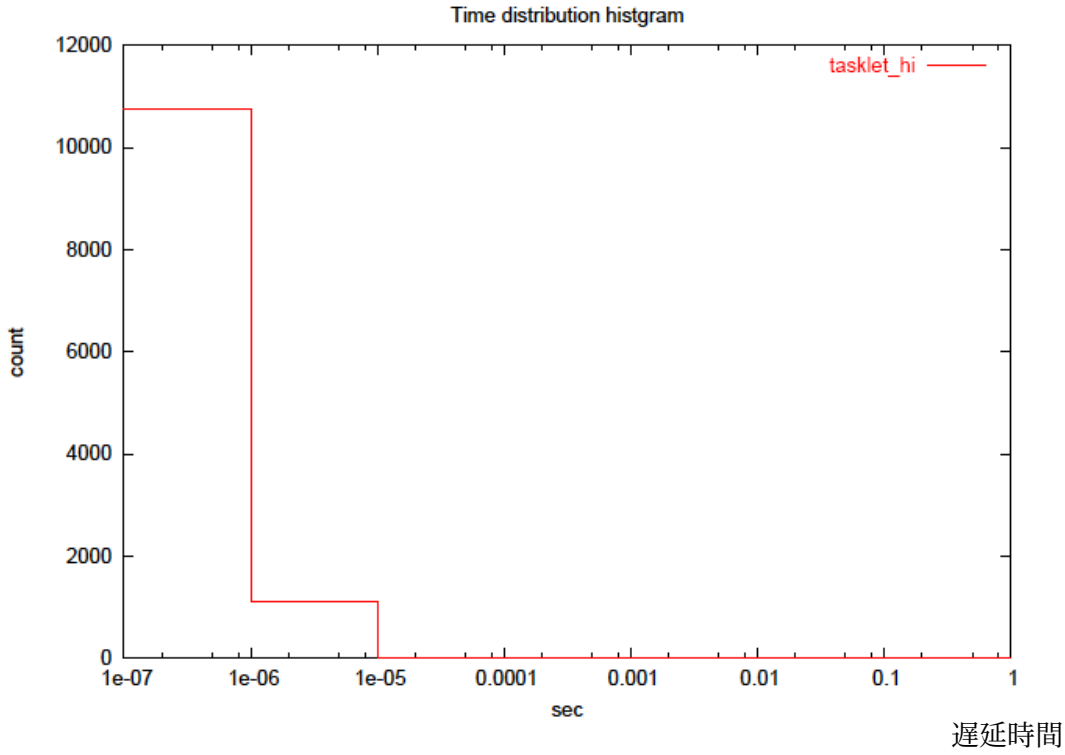

図 3.1-2 tasklet hi のソフトウェア割り込みの実行遅延時間の対数分布

図 3.1-2 は、横軸が遅延時間を表す。それぞれの項目が、

(1)1e-07　0.1 マイクロ秒以下

(2)1e-06　0.1 マイクロ秒～1 マイクロ秒

(3)1e-05　1 マイクロ秒～10 マイクロ秒

(4)10 マイクロ秒～100 マイクロ秒

というように、計測された遅延時間の幅を表し、縦軸が記録された回数を表す。

図 3.1-2 の測定結果より、タスクレットのソフトウェア割り込みの 99.9%が 10 マイクロ秒以内に実行されていることが分かる。

このように LKST 性能評価機能を利用することで、カーネルのソフトウェア割り込みの遅延時間の情報を取得し、解析できることが確認できた。

3.1.4. メモリ関連処理時間

LKST 性能評価機能を利用して、カーネル内のメモリ確保・メモリ解放処理に関する情報を取得できることを確認するため、カーネルビルド時のデータを解析した。

カーネルビルド時のメモリ確保処理時間に関する統計情報を、次のコマンドで取得したものを表 3.1-5 に示す。

```
# lkstla palloc -s sebuf00.0 > palloc.stat
```

表 3.1-5 ページ確保処理に要した時間の統計情報

order	allocation-size	count	average	max	min
0	1pages	120965	0.0000004	0.000068058	0.000000159
1	2pages	1338	0.000001544	0.000009366	0.000000711

表 3.1-5 は、左側からそれぞれ次の意味を表す。
(１)ページ確保要求のサイズ(2 の累乗で表される)
(２)実際に確保されるページの数(IA-32 ハードウェアでは、1page あたり 4KB)
(３)記録された回数
(４)ページ確保処理に要した時間の平均
(５)ページ確保処理に要した時間の最大
(６)ページ確保処理に要した時間の最小

同じ計測データをもとに、1 ページの確保要求が行われた際のページ確保処理時間を、lkst_plot_log を利用して時系列のグラフに変換したものを、図 3.1-3 に示す。

```
# lkstla palloc -l sebuf00.0 > palloc.log
# lkst_plot_log palloc.log 0
```

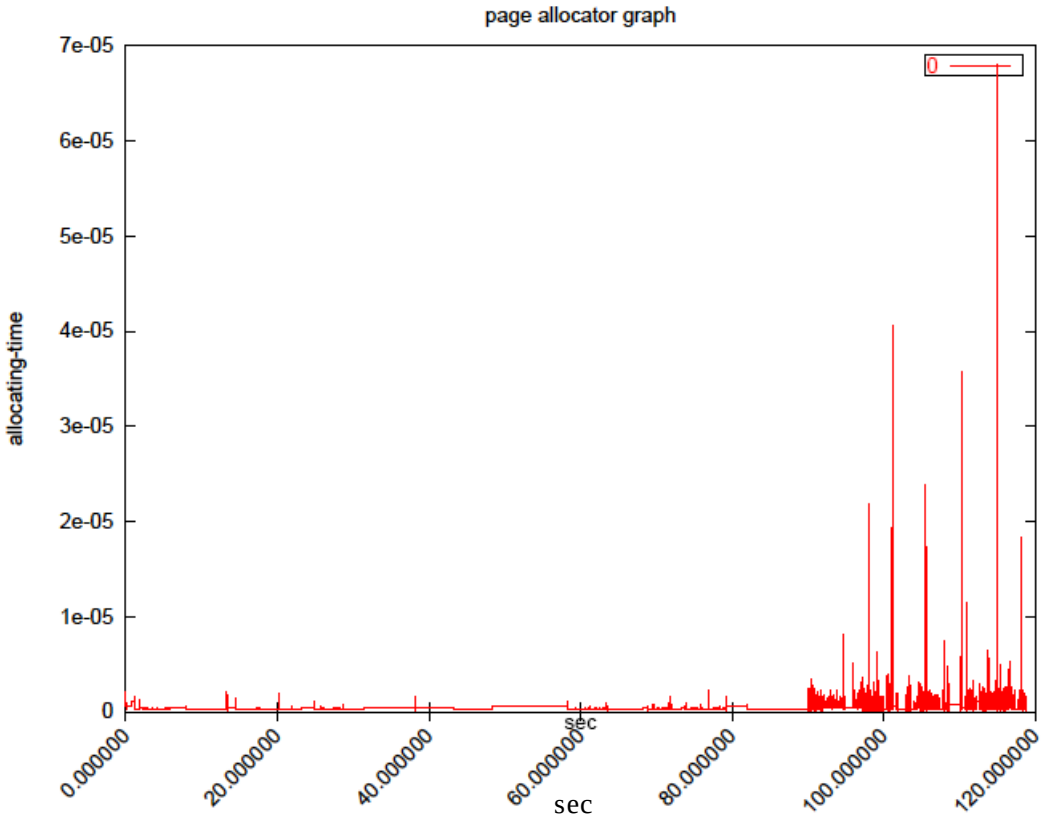

図 3.1-3 1 ページの確保に要した時間の時系列データ

図 3.1-3 は横軸がイベントが記録された時刻、縦軸がページ確保処理の時間を表している。

図 3.1-3 より、取得したログの後半部分で、ページ取得に要する時間が長くなっていることが分かる。そこで、次のコマンドで、同じ計測データからページの解放処理に要した時間のデータを取得した。

```
# lkstla vmscan sebuf00.0 > vmscan.log
```

このデータをもとに、ページ解放処理が記録された時刻を、図 3.1-3 と同じ時系列で重ね合わせてグラフ化したものを図 3.1-4 に示す。

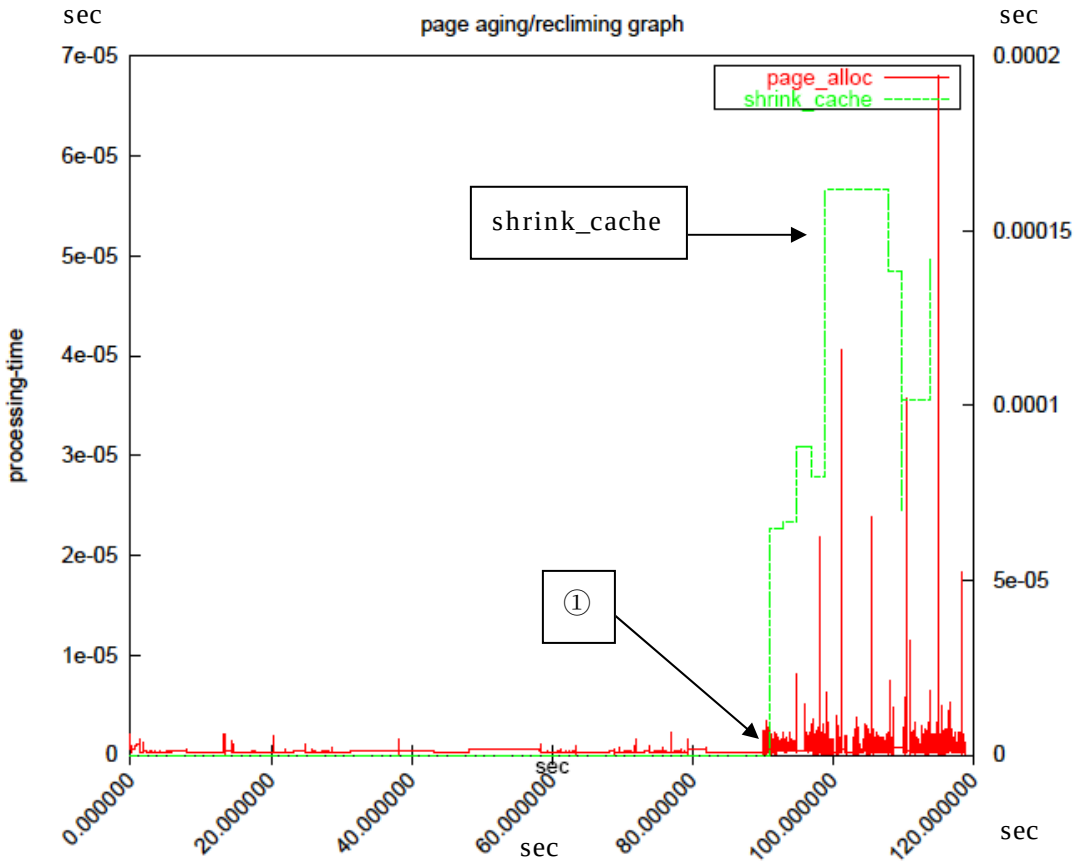

図 3.1-4 ページ解放処理(shrink_cache)の実行状況の時系列データ

図 3.1-4 は、横軸はイベントが記録されている時刻を表し、左の縦軸がページ確保処理時間、右の縦軸がページ解放処理時間を表している。ページ確保処理時間と比べて、ページ解放処理時間がかかるため、それぞれのグラフで表される時間のスケールを変更してある。

図 3.1-4 より、カーネルビルドの開始時には非常に短い時間でメモリ確保処理が行われていることがわかる。この時点ではシステムに空きメモリが十分あったため、メモリ確保要求を行うと、空きページがすぐに割り当てられたためである。

しかし①の時点からページ解放処理のイベント(shrink_cache)の記録が開始され、このタイミングにあわせてページ確保処理時間も長くなっていることがわかる。

これは、kernel の実装としてページ確保時に空きページの数がある閾値より下回っていると、ページを確保するためにページ解放処理が実行されるためである。

このように、LKST 性能評価機能を利用することで、カーネルのページ確保、ページ解放処理の情報を取得し、解析できることが確認できた。

3.1.5. ブロック I/O

LKST 性能評価機能を利用して、カーネル内のブロック I/O 処理の情報を取得・解析できることを確認するために、カーネルビルド時のデータを解析した。

I/O リクエスト処理時間に関する統計データを、表 3.1-6 に示す。

```
# lkstla biotime -s sebuf00.0 > biotime.stat
```

表 3.1-6 デバイスに対する I/O 要求処理時間の統計

type	kdevice-rw	count	average	max	min
801	00000801-RD	836	0.003109783	0.084165987	0.000089993
803	00000803-RD	15783	0.001929313	0.58819506	0.000086418
80000801	00000801-WR	225	0.275993157	5.48464832	0.000329315
80000803	00000803-WR	1622	0.528497976	24.11893303	0.00080076

表 3.1-6 は、左側の項目よりそれぞれ次の意味を表す。

(1)ブロックデバイス
(2)I/O リクエストの種類
(3)記録された回数
(4)I/O リクエストの要求処理時間の平均
(5)I/O リクエストの要求処理時間の最大
(6)I/O リクエストの要求処理時間の最小

00000803-RD は、major 番号 8、minor 番号 3 のデバイスに対する読み込みを表す。
00000803-WR は、major 番号 8、minor 番号 3 のデバイスに対する書き込みを表す。
したがって該当するデバイスは、/dev/sda3 となる。同様に 00000801-RD/WD は、/dev/sda1 に対する読み込み・書き込みを表す。

/dev/sda3 に対する読み込みと書き込み処理時間の対数分布を

図 3.1-5 と図 3.1-6 に示す。

```
# lkstla biotime -d sebuf00.0 > biotime.dist
# lkst_plot_log biotime.dist 803
# lkst_plot_log biotime.dist 80000803
```

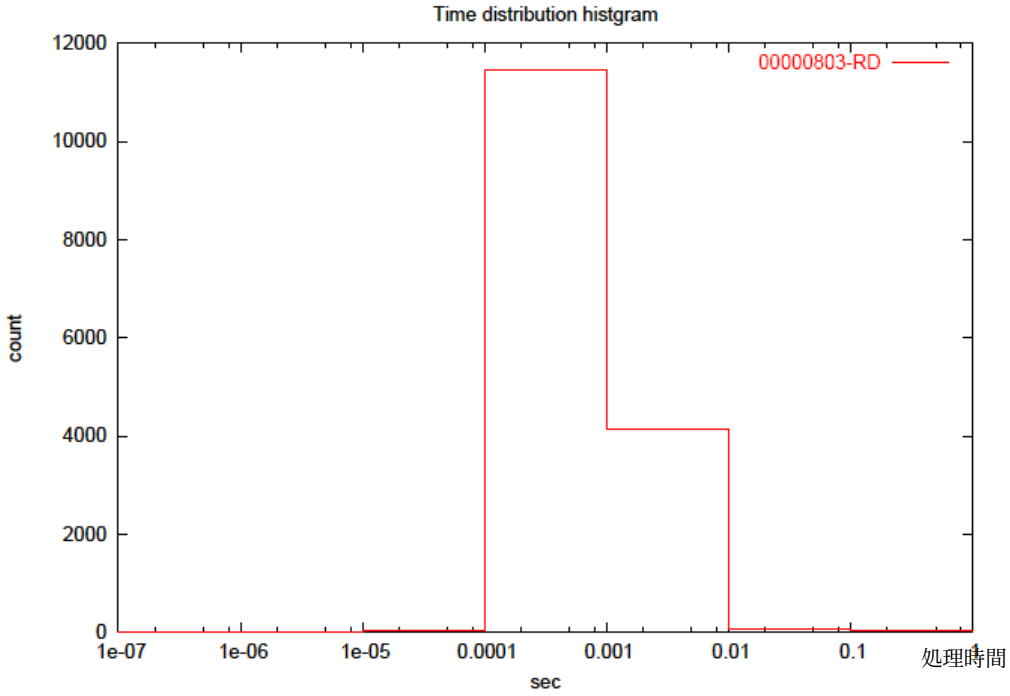

図 3.1-5 読み込みの I/O リクエスト処理時間の分布

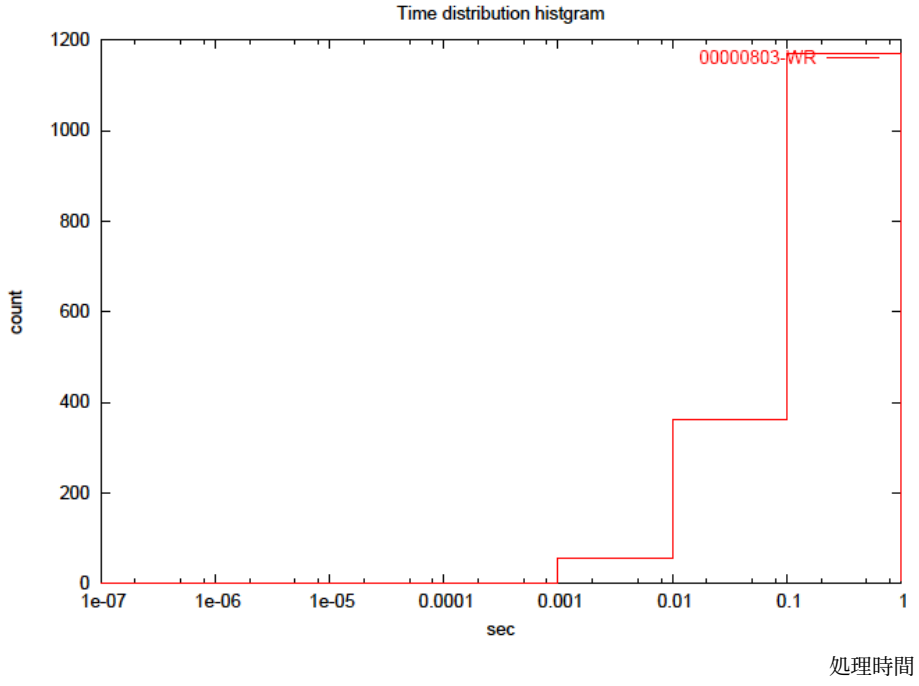

図 3.1-6 書き込みの I/O リクエスト処理時間の分布

図 3.1-6 の結果から、読み込み要求は 10 マイクロ秒～100 ミリ秒(平均 2 ミリ秒)で処理されるのに対し、書き込み要求は 100 マイクロ秒～10 秒(平均 528 ミリ秒)で処理されている。

このとき/dev/sda の I/O リクエストキューに投入されているリクエスト数の統計情報を次のコマンドで取得した。

```
# lkstla blkqueue -s sebuf00.0 > blkqueue.stat
```

取得したデータを、表 3.1-7 に示す。

表 3.1-7 I/O リクエストキューに投入されている I/O リクエスト数の統計情報

rq-addr	requestqueue	count	average	max	min
f7967e18	0xf7967e18RD	9756	1.005330053	4	1
f7967e19	0xf7967e18WR	744	37.51478495	132	1

表 3.1-7 は、左側の項目からそれぞれ次の意味を表す。

(1)I/O リクエストが登録されているキューのアドレス
(2)I/O リクエストキューのアドレスとリクエストの種別
(3)イベントが記録された回数
(4)I/O リクエストキューの平均 I/O リクエスト数
(5)I/O リクエストキューの最大 I/O リクエスト数
(6)I/O リクエストキューの最小 I/O リクエスト数

I/O リクエストキューのアドレスは、マシン起動時の dmesg に次のように表示される。

```
SCSI subsystem driver Revision: 1.00
scsi1 : Adaptec AIC79XX PCI-X SCSI HBA DRIVER, Rev 2.0.12.AX1
        <Adaptec AIC7902 Ultra320 SCSI adapter>
        aic7902: Ultra320 Wide Channel B, SCSI Id=7, PCI-X 67-100Mhz, 512 SCBs

blk: queue f7967e18, I/O limit 4095Mb (mask 0xffffffff)
(scsi1:A:1): 320.000MB/s transfers (160.000MHz DT|IU|QAS, 16bit)
  Vendor: FUJITSU     Model: MAP3367NC            Rev: 0108
  Type:   Direct-Access                           ANSI SCSI revision: 03
```

また、I/O リクエストキューに投入されている I/O リクエスト数の推移を確認するために、時系列データを取得し、図 3.1-7、図 3.1-8 に示す。

```
# lkstla blkqueue sebuf00.0 > blkqueue.log
# lkst_plot_log blkqueue.log f7967e18
# lkst_plot_log blkqueue.log f7967e19
```

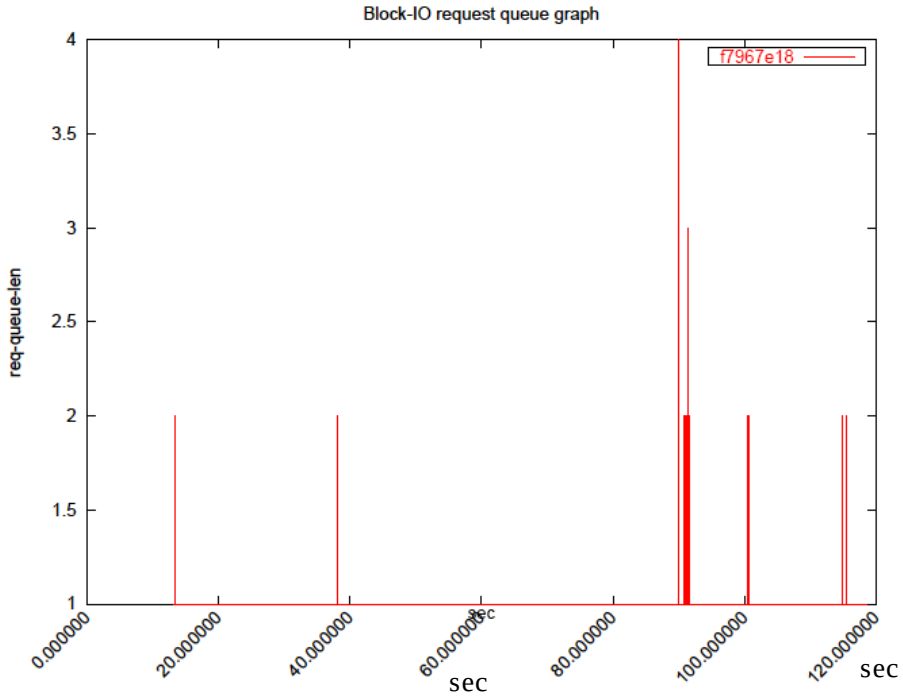

図 3.1-7 読み込みの I/O リクエスト数の推移

図 3.1-7 は、横軸はイベントが記録された時刻、縦軸は I/O リクエストキューに登録されていた I/O リクエスト数を表す。

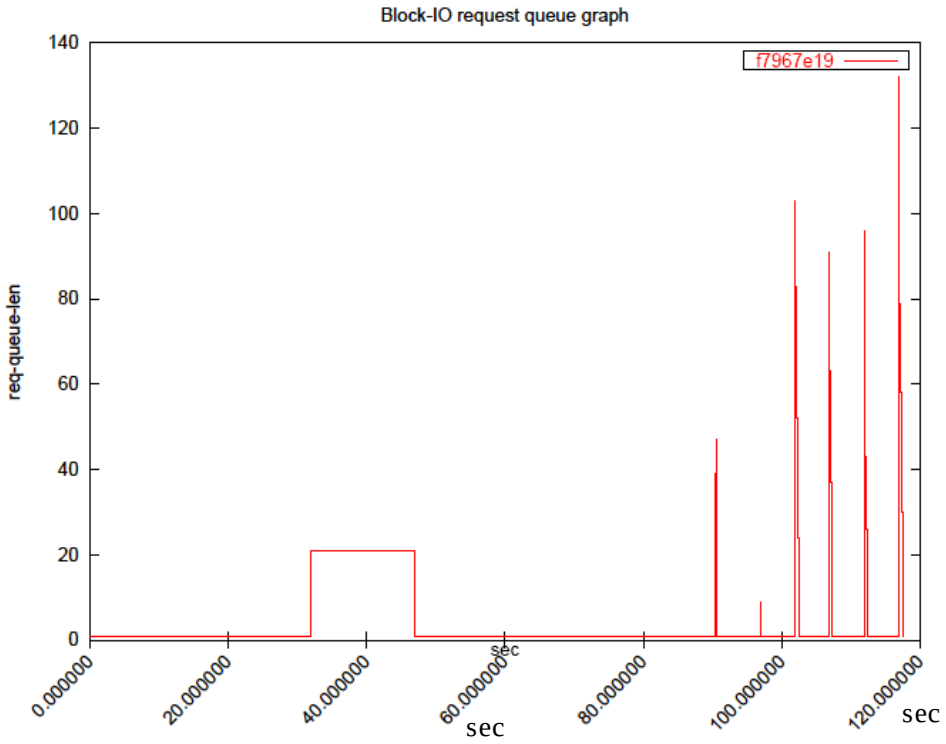

図 3.1-8 書き込みの I/O リクエスト数の推移

図 3.1-8 の書き込み I/O リクエストキューの状態と、図 3.1-4 のページ解放処理の状態をあわせて見ることで、ページ解放にあわせてキャッシュの書き戻しが行われた結果、ディスクへデータを書き出すための I/O 要求が多数発生していることがわかる。

3.1.6. I/O 処理が高負荷時の解析

カーネルビルドが完了し、LKST によって記録されたデータをファイルに書き込む処理によって、書き込み処理が非常に頻繁に行われていた時点のデータの解析も行った。

図 3.1-9 に書き込み処理に要した時間を時系列のグラフとして示す。

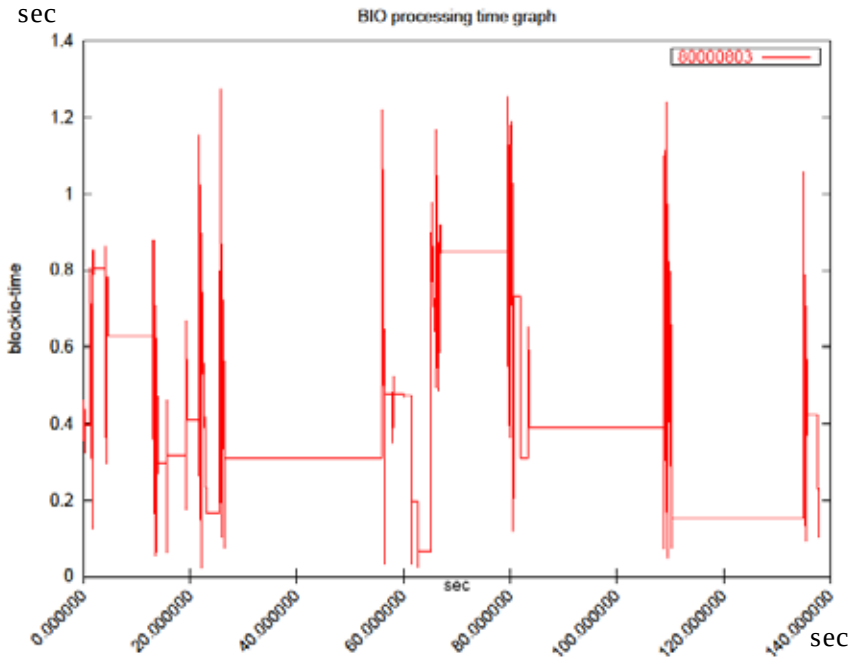

図 3.1-9　書き込み処理に要した時間の時系列データ

同じデータから I/O リクエストキューに投入されていた I/O リクエスト数の時系列データとして図 3.1-10 のデータを得た。

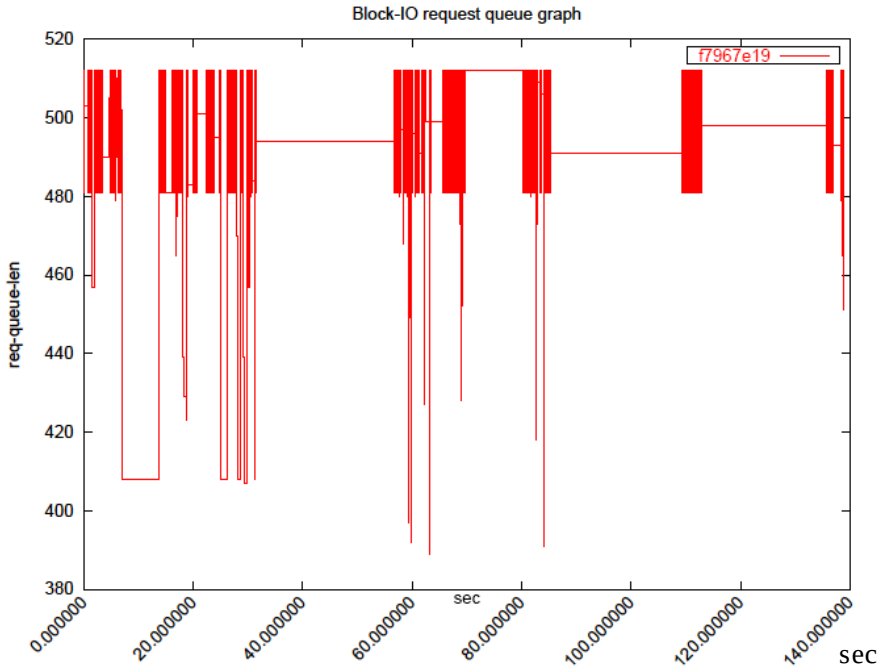

図 3.1-10 I/O リクエストキューの I/O リクエスト数の推移

MIRACLE LINUX V3.0 の I/O リクエストキューには、READ、WRITE それぞれ 512 個のリクエストを投入することができる。図 3.1-10 は WRITE の I/O リクエストキューの状態を表しており、平均で 494 個の I/O リクエストが投入されていて、常に WRITE 処理が行われていたことがわかる。

図 3.1-9 の WRITE 要求の処理時間と、図 3.1-10 の I/O リクエストキューの状態を付き合わせると、I/O リクエストキューが満杯の状態のときに、WRITE 要求の処理時間が伸びていることがわかる。これは、I/O リクエストキューが満杯のために、I/O リクエストをキューに投入できずにキューが空くのを待った結果、処理時間が伸びたと考えられる。

以上のように、LKST 性能評価機能を利用して、ブロック I/O 処理に関する情報の取得・解析ができることを確認した。

3.1.7. スケジューラ

LKST 性能評価機能を利用して、カーネルのスケジューリング機能に関する情報の取得・解析ができることを確認するために、chat ベンチ実行中のデータを解析した。

chat ベンチ実行中のランキューのプロセス数とスケジューラのオーバヘッドの関係を確認するために、LKST 性能評価機能を利用して、スケジューリングに要する時間を測定した。

```
# lkst schedrun -s sebuf00.0 > schedrun.stat
# lkst_plot_stat schedrun.stat
```

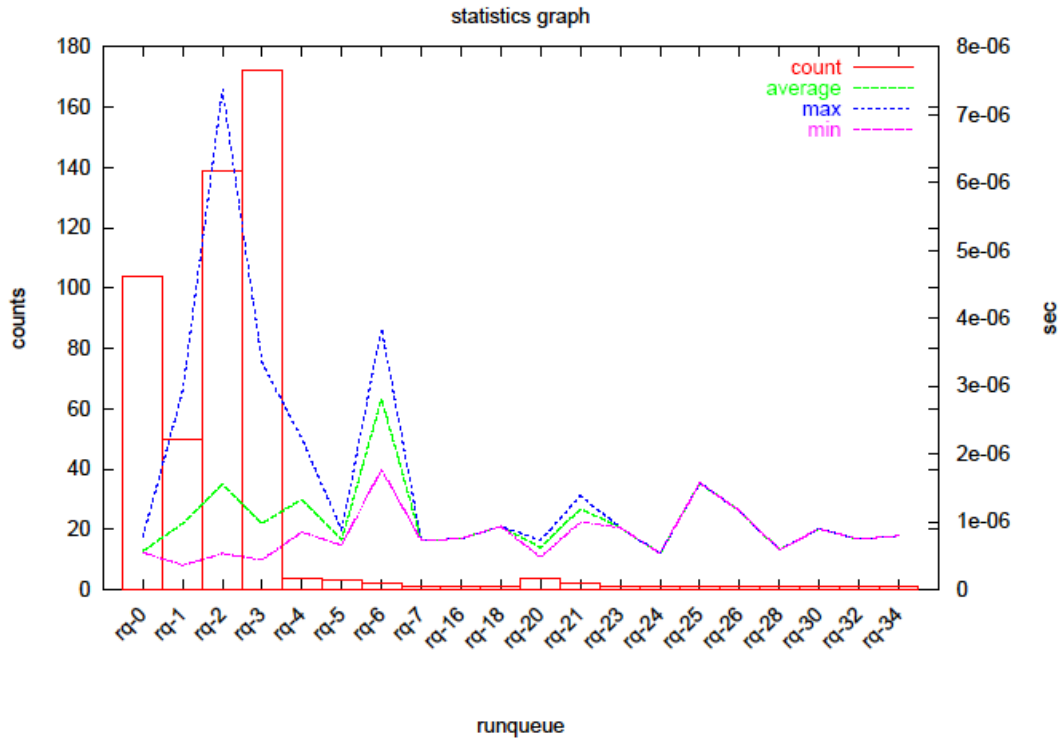

図 3.1-11 ランキューのプロセス数とスケジューリング時間の関係

図 3.1-11 は、横軸がランキューに登録されているプロセス数を表し、縦軸(折れ線)がスケジューリングに要した時間を表す。折れ線で表される平均スケジューリング時間を確認すると、ランキューに登録されているプロセス数が多くなっても、スケジューリング時間は増加しないことが分かる。

カーネル 2.4 のスケジューラは、プロセス数の増加に伴ってスケジューリング時間が増大する問題点が既に広く知られているが、MIRACLE LINUX V3.0 には O(1)スケジューラと呼ばれるカーネル 2.6 で採用されたスケジューラがバックポートされている。

O(1)スケジューラは、プロセス数が増加してもスケジューリング時間が一定であるという特長をもっており、LKST 性能評価機能を利用することで O(1)スケジューラの効果を明確に捉えることができた。このような特性はこれまでベンチマークの結果と考察などから確認は行われていたが、LKST 性能評価機能を利用することで、カーネル内部のイベントを厳密に捉えて、性能を評価することが可能になった。

以上のように、LKST 性能評価機能を利用することでカーネルのスケジューラに関する情報の取得・解析ができることが確認できた。

3.1.8. スピンロック

LKST 性能評価機能を利用して、カーネルのスピンロックに関する情報の取得・解析ができることを確認するために、chat ベンチ実行中のデータを解析した。

chat ベンチ実行中のスピンロックによるロック占有時間を次のコマンドで取得し、取得したデータを表 3.1-8 に示す。

```
# lkstla spinlock -s sebuf00.0 > spinlock.stat
```

表 3.1-8 スピンロックの占有時間

address	calling_point	count	average	max	min	累積占有率
c012ae73	schedule_Rsmp_4292364c+6e3	446	0.004129600	0.499988939	0.000000322	77.0%
f888cb97	_insmod_ext3_S.text_L7+4b37	7783	0.000022955	0.000250711	0.000019686	84.5%
f888dcf8	_insmod_ext3_S.text_L7+5c98	7806	0.000016023	0.000206990	0.000012449	89.7%
c017355f	unregister_cache_Rsmp_9+535f	34308	0.000000793	0.000238073	0.000000189	90.9%
c018d4c2	try_to_free_buffers_Rsm+442	14943	0.000001590	0.000211208	0.000000173	91.9%
f887f7bb	journal_blocks_per_page+7db	101456	0.000000175	0.000070043	0.000000132	92.6%
f887fba3	journal_blocks_per_page+bc3	101459	0.000000170	0.000073940	0.000000133	93.3%
f8876181	journal_unlock_updates_+881	93639	0.000000173	0.000058926	0.000000133	94.0%
f8876da1	journal_dirty_metadata_+251	62434	0.000000232	0.000095910	0.000000178	94.6%
c016debf	kmem_find_general_cache+259f	15600	0.000000532	0.000071986	0.000000285	95.0%

スピンロック占有の影響の影響度を count×average の大きさで判断した。得られた 133 項目のデータから、この値が大きいものを上位から 10 項目抜き出し、表 3.1-8 に示す。

表 3.1-8 は左側の項目から順に次の意味を表す。

(1)記録されたスピンロックのアドレス
(2)スピンロックが呼び出されている関数
(3)イベントが記録された回数
(4)スピンロックによってロックしていた平均時間
(5)スピンロックによってロックしていた時間の最大時間
(6)スピンロックによってロックしていた時間の最小時間
(7)スピンロック全体における影響度を把握するための累積占有率（なお、この項目だけは解析ツールでは算出されないため、表計算ソフトによって計算した。）

次に、スピンロックを取得するために待たされている状況を表す busywait の情報も同じデータから次のコマンドで取得した。

```
# lkstla busywait -s sebuf00.0 > busywait.stat
```

busywait の影響度を count×average の大きさで判断した。得られた 132 項目のうち、この値が大きいものを上位から 10 項目抜き出し、表 3.1-9 に示す。

表 3.1-9 busywait の統計情報

address	calling_point	count	average	max	min	累積占有率
f888d7a5	_insmod_ext3_S.text_L7+5745	7806	0.000009928	0.000109401	0.000000132	24.3%
f888dcf8	_insmod_ext3_S.text_L7+5c98	7809	0.000006272	0.001264286	0.000000133	39.7%
f888d6c6	_insmod_ext3_S.text_L7+5666	7809	0.00000557	0.001422885	0.000000131	53.3%
f88779ec	journal_try_to_free_buf+10c	18687	0.000002284	0.038224219	0.000000101	66.7%
f887fba3	journal_blocks_per_page+bc3	101461	0.000000164	0.000229349	0.000000128	71.9%
f887f7bb	journal_blocks_per_page+7db	101458	0.000000161	0.000072111	0.000000127	77.1%
f8876181	journal_unlock_updates_+881	93642	0.000000158	0.000068566	0.000000129	81.7%
f8876da1	journal_dirty_metadata_+251	62434	0.000000159	0.000058116	0.00000013	84.8%
c017355f	unregister_cache_Rsmp_9+535f	34309	0.000000186	0.00004472	0.000000101	86.8%
c012b2ce	_wake_up_Rsmp_127fda83+12e	29955	0.000000165	0.000001639	0.000000103	88.4%

これらの 2 つの表を比べると、

(１)0xf888dcf8　__insmod_ext3_S.text_L7+5c98
(２)0xf887fba3　journal_blocks_per_page+bc3
(３)0xf887f7bb　journal_blocks_per_page+7db
(４)0xf8876181　journal_unlock_updates_+881
(５)0xf8876da1　journal_dirty_metadata_+251
(６)0xc017355f　unregister_cache_Rsmp_9+535f

といったアドレスが共通に存在しており、これらの処理に関しては、スピンロックの占有時間が長いために、他のプロセスからのスピンロック要求を待たせていると考えられる。特に、EXT3 に関連したジャーナリング処理が数多く計測されていることがわかり、カーネル 2.4 のジャーナリング処理には改善の余地がある可能性が高い。

これまでロック競合を検出するためには、カーネルのソースコードからロック競合箇所を推測するか、lockmeter[1]と呼ばれるロック競合の検出のためのパッチを適用したカーネルでロック競合を探し出す必要があった。ところが今回開発した LKST 性能評価機能を利用すると、ロック競合の情報を正確に、かつ簡単に取得することができ、Linux

[1] http://oss.sgi.com/projects/lockmeter/

カーネルのロック競合を解消するための調査に大いに役立つことが予想される。

以上のように、LKST 性能評価機能を利用して、カーネルのスピンロックに関する情報の取得・解析ができることを確認した。

3.1.9. ウェイトキュー

LKST 性能評価機能を利用して、ウェイトキューの情報の取得・解析ができることを確認するため、chat ベンチ実行中のウェイトキューの情報を解析した。

```
# lkstla waitqueue -s sebuf00.0 > waitqueue.stat
```

表 3.1-10 にウェイトキューに登録された時点で、そのウェイトキューに登録されていたプロセス数を示す。

表 3.1-10 ウェイトキューに登録されていたプロセス数の統計情報

address	calling_point	count	average	max	min
c013b6ba	sys_wait4_Rsmp_a2c21821+8a	162	0	0	0
c014f4d6	schedule_task_Rsmp_2d6c+2a6	3	0	0	0
c015f3b8	__wait_on_page_Rsmp_55+98	4	0	0	0
c01730c7	unregister_cache_Rsmp_9+4ec7	4718	0.000847817	1	0
c0173192	unregister_cache_Rsmp_9+4f92	43	0	0	0
c01872c8	__wait_on_buffer_Rsmp_5+38	1	0	0	0
c02701de	__lock_sock_Rsmp_e32bf2+5e	65	0.138461538	1	0
c02a99d5	tcp_ioctl_Rsmp_532adb73+25e5	42	2.880952381	11	0
c02ac517	cleanup_rbuf_Rsmp_2a4af+197	308	0.012987013	4	0
c02aee11	tcp_disconnect_Rsmp_9a8+481	22	0	0	0
c02d63b7	inet_dgram_connect_Rsmp+e7	19	0	0	0
f887d588	log_wait_commit_Rsmp_c8+78	1	0	0	0
f89d1934	lkst_buffer_read_Rsmp_d+1684	7	0	0	0

表 3.1-10 は、左側の項目から順に次の意味を表す。

(1)ウェイトキューへの登録処理を行った命令アドレス
(2)ウェイトキューへの登録処理を行った関数
(3)イベントが記録された回数
(4)登録時のウェイトキューの長さの平均

(５)登録時のウェイトキューの長さの最大
(６)登録時のウェイトキューの長さの最小

また、それぞれのウェイトキューでの待ち時間のデータを表 3.1-11 に示す。

```
# lkstla waittime -s sebuf00.0 > waittime.stat
```

表 3.1-11 ウェイトキューでの待ち時間の統計情報

address	calling_point	count	average	max	min
c02ac517	cleanup_rbuf_Rsmp_2a4af+197	215	0.151444268	0.696683088	0.002870083
c02a99d5	tcp_ioctl_Rsmp_532adb73+25e5	22	0.223851876	0.343509411	0.042565431
c02701de	__lock_sock_Rsmp_e32bf2+5e	48	0.053467565	0.294112076	0.000001307
c0173192	unregister_cache_Rsmp_9+4f92	42	0.021716285	0.247938042	0.000026528
c014f4d6	schedule_task_Rsmp_2d6c+2a6	2	0.439507328	0.534058003	0.344956654
f89d1934	lkst_buffer_read_Rsmp_d+1684	6	0.065204266	0.08674661	0.028490241
c02aee11	tcp_disconnect_Rsmp_9a8+481	22	0.008198285	0.179596473	0.000022973
c015f3b8	__wait_on_page_Rsmp_55+98	3	0.00468667	0.011018849	0.001321996
c01730c7	unregister_cache_Rsmp_9+4ec7	4718	0.000000954	0.000205985	0.000000625
c013b6ba	sys_wait4_Rsmp_a2c21821+8a	162	0.000005201	0.00003339	0.000000428
c02d63b7	inet_dgram_connect_Rsmp+e7	18	0.000012876	0.000018898	0.000010971

表 3.1-11 は、左側の項目から順に次の意味を表す。
(１)ウェイトキューへの登録処理を行った命令アドレス
(２)ウェイトキューへの登録処理を行った関数
(３)イベントが記録された回数
(４)ウェイトキューで待ち状態であった平均時間
(５)ウェイトキューで待ち状態であった時間の最大
(６)ウェイトキューで待ち状態であった時間の最小

表 3.1-11 は、ウェイトキューで待っている状態が多い処理を選び出すために、ウェイトキューに登録された回数(count)×平均時間(average)の上位 10 項目を降順に並べている。この結果から、非常に頻繁に呼び出される unregister_cache()内のウェイトキューよりも、cleanup_rbuf()や、tcp_ioctl()のウェイトキューの処理のほうが chat ベンチ実行時には影響が大きいといえる。

以上のように、LKST 性能評価機能を利用して、カーネルのウェイトキューの情報を取得・解析できることを確認した。

3.2. OS 層評価における利用

OS 層の限界調査において、今回開発した LKST 性能評価機能を利用した。
詳細については、「OS 層の評価 3.2 Iozone による評価」を参照のこと。

3.3. システムコール性能の劣化解析での利用

IO 処理プログラム実行中に、どのようなシステムコールがどれくらい実行されているのか、時間のかかっているものは何か、などを、LKST を使って情報を収集し、lkstla を使って評価・分析する。

3.3.1. 評価環境

（1）システム環境

評価を行ったシステム環境を表 3.3-1 に示す。

表 3.3-1 評価を行うシステム環境

項番	項目		評価環境
1	ハードウェア	CPU	Pentium4 3.08GHz Hyper Threading 有効
2		メモリ	1GByte
3		ハードディスク	120GB ATA
4	ソフトウェア	ディストリビューション	Fedora Core 2
5		カーネル	upstream 2.6.9+新 LKST
6		解析ツール	lkstla
7		負荷ツール	IO 処理プログラム(20MB のデータを DIRECT IO を使って書き込むプログラムを 6 並列で動作)

（2）LKST 実行条件

評価を行う際の、LKST の実行条件は以下の通り。

(a) 採取するイベントはシステムコールの入り口と出口及びブロック IO 関連のイベント

(b) 使用するハンドラは DEFAULT のみ

(c) 記録バッファはサイズ 10MB を 1 個使用

(d) LKST は負荷ツール実行直前にトレースを開始、負荷ツール終了で結果を出力

3.3.2. 評価手順

（1） LKST の今回開発した機能を使って、性能データを収集する

(a) LKST を実行開始する

(b) lkstm コマンドで、3.3.1(2) (a)(b)の内容を指定した評価用マスクセットを作成し有効にする

(c) lkstbuf コマンドで、記録バッファ(3.3.1(2)(c))を作成し、書き込み先に指定する
(d) LKST をいったん停止する
(e) 評価用スクリプト(3.3.1(2)(d)の内容)を作成する。
（上記で評価準備完了）
(f) (e)のスクリプトを実行し、データを記録する
(g) lkstbuf コマンドを用いて、収集したデータをファイル(logdata)へ格納する

（２） 同じく今回開発した解析ツール lkstla を使って LKST データを解析する
(a) 各システムコールの実行回数と最大／最小／平均処理時間を抽出し、可視化する
(b) 各システムコールの処理時間の分布を可視化する

3.3.3. 結果と考察

3.3.2(2)(a)の結果を図 3.3-1 に示す。

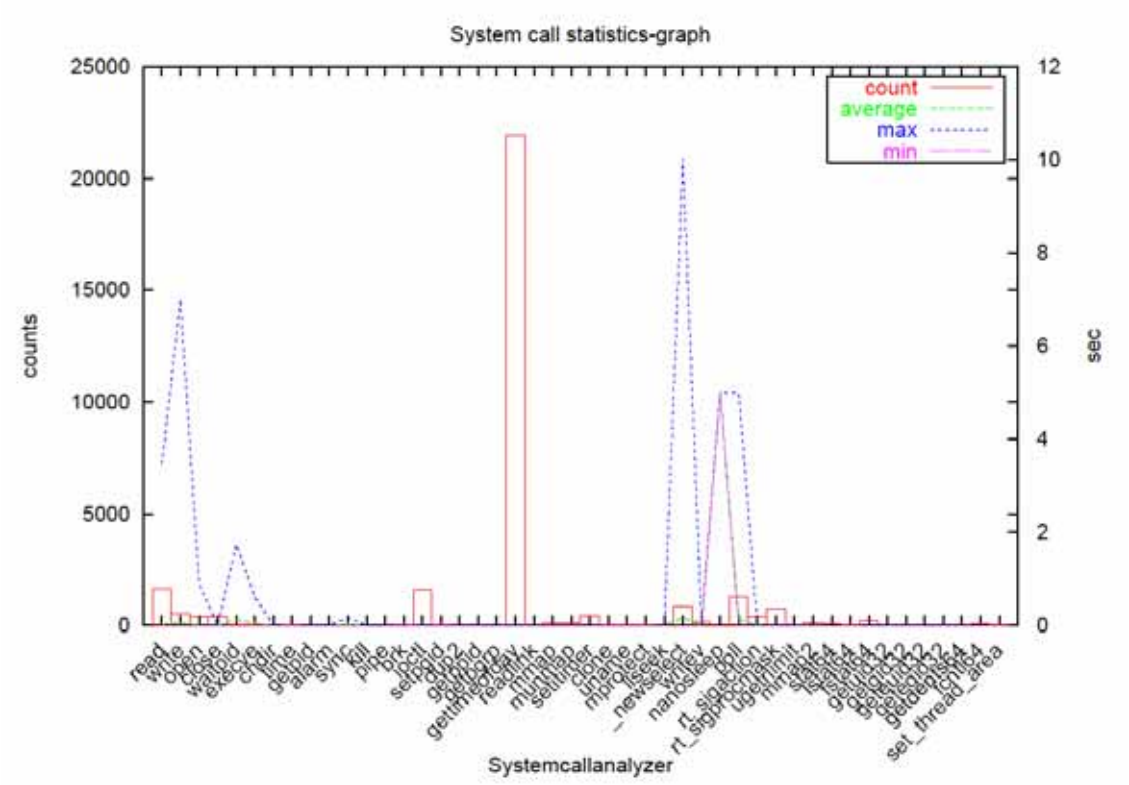

図 3.3-1 各システムコールの実行回数と処理時間の最大／最小／平均

最大処理時間が多いのは wait 関係、sync など、条件次第で時間がかかるものであり、一見、問題なさそうに見える。しかし、(b)を実施し、個々のシステムコールの処理性能に関して分析してみると、write システムコールで図 3.3-2 のような処理時間の分布が得られた。

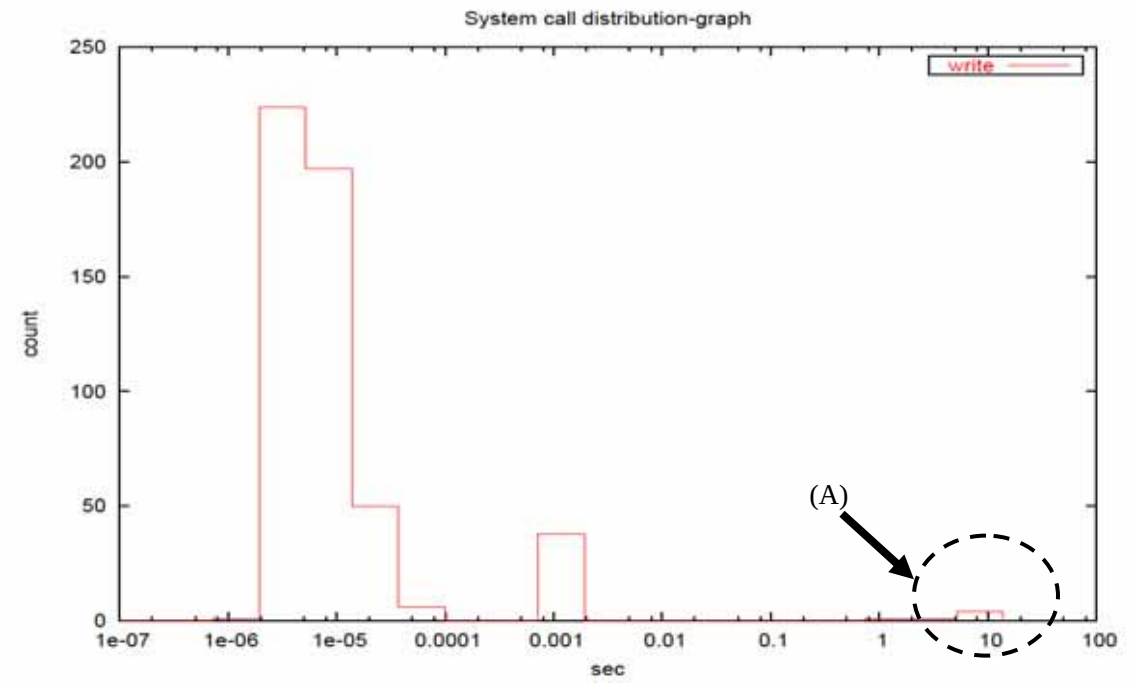

図 3.3-2 write システムコールの処理時間の分布

この分布図から、数回であるが write システムコールが遅い時がある(図中(A))。

そこで、この処理が発生している時にカーネルで何が起きているかを調査する。

処理に時間がかかっている write システムコールを特定するために、以下のコマンドを実行して write のログだけを取り出し、処理時間をキーにしてソートを行った。

```
# lkstla syscall -k4 -l  logdata | sort -k 4 -n
```

この結果、図 3.3-3 に示す結果が得られた。

```
System call analyzer
sysno       syscall_name           start[sec] processing-time
    4              write 1108124728.994415926     0.000001596
    4              write 1108124738.190588650     0.000002207
    4              write 1108124737.619677569     0.000002313
    4              write 1108124737.619730072     0.000002383
                                 ......
    4              write 1108124756.704982197     0.001472200
    4              write 1108124756.684876630     0.001854759
    4              write 1108124740.386500601     1.615312588
    4              write 1108124741.025591443     3.212324512
    4              write 1108124741.491330318     5.896216290
    4              write 1108124741.417396792     6.355426504
    4              write 1108124741.648947090     6.658438593
    4              write 1108124741.508154420     7.009370341
```

図 3.3-3 write システムコールの処理時間

遅くなっている処理はある特定の時間帯（1108124740.386500601～1108124748.577504761）に集中していることがわかる。この時間帯に実行されたIO要求（Block IOの処理）の処理時間の分布を調べるため、以下のコマンドを実行した。

```
# lkstla biotime -d::50: -t1108124740.386500601:1108124748.577504761 logdata
```

結果のグラフを以下に示す。

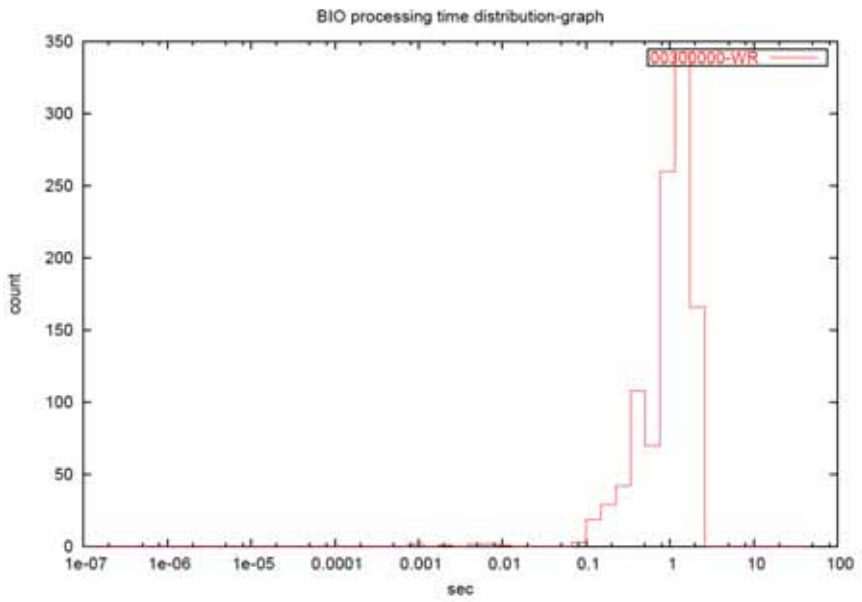

図 3.3-4 write IO要求処理時間の分布（処理低下時）

Block IOの処理完了までの時間が、ほとんどは1秒以上かかっており、2秒以上かかっているものも多数あることが確認できる。

一方、この時間帯以外の時間に実行された IO 要求の処理時間の分布を調べるため、以下のコマンドを実行した。

```
# lkstla biotime -d::50: -t¥!1108124740.386500601:1108124748.577504761 logdata
```

結果のグラフを図 3.3-5 に示す。

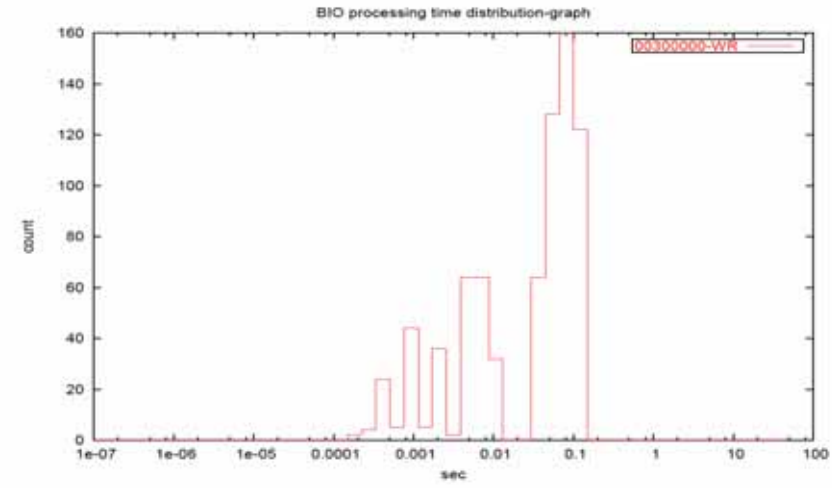

図 3.3-5 write IO 要求処理時間の分布（通常時）

ほとんどの IO 処理が 0.1 秒以下で終了していることがわかる。

これらのグラフを比較すると、システムコールの処理時間の遅延が、IO 処理そのものの遅延によるものであることがわかる。次に、IO 処理がどうして遅くなるのかについて、IO 処理要求のキューの長さを調べた。

```
# lkstla blkqueue -l logdata
```

この結果、得られたグラフを調整し、問題の時間帯の開始が 0 になるようにしたグラフを図 3.3-6 に示す。

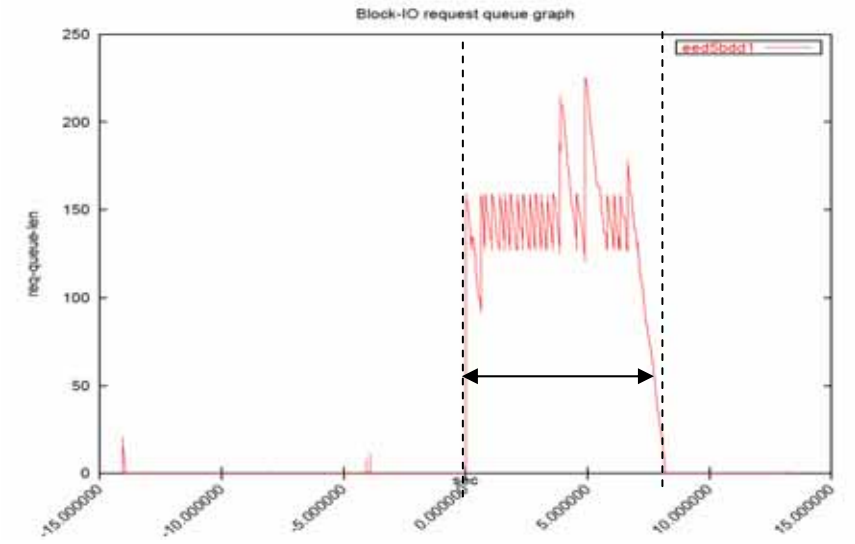

図 3.3-6 write IO 要求キューの長さの分布（通常時）

このグラフから、システムコールの処理が遅くなっている時間帯（矢印部分）では、IO 処理要求キューが非常に長くなっており、処理が混み合っていることがわかる。
これらのことから、IO 処理要求キューが一定以上の長さに達すると、IO 処理性能が低下し、それにつられてシステムコールの性能が極端に遅くなると判断できる。

このように、LKST を利用した性能評価ツール lkstlogtools により、システム性能の劣化について以下の解析が可能となる。

(1) 特定のシステムコールの性能が劣化しているかどうかを調べることが可能
(2) 劣化している場合、その時間帯と通常の時間帯について、システムコール毎に関連情報のログを解析することが可能
(3) 劣化の時間帯の情報をキーにカーネルのキュー状態を分析し、因果関係の有無を調べることが可能
(4) 劣化の時間帯の情報をキーに他のカーネル状態を取得できれば、上記状態との因果関係の有無を調べることが可能（他のカーネル状態の取得には別機能を利用する必要があるため、今回は未調査）

これらの情報は性能劣化の原因を特定する上で有効であり、特定までの時間を短縮することが出来ると考える。
但し、これ以上の原因の特化や解決案に関しては、他の機能で情報収集したりカーネルのソースコードを解析したりするなど、他の手段と組み合わせて行なう必要があると考える。

3.4. LKST 性能評価機能のオーバヘッド調査

LKST 性能評価機能を組み込んだことによるカーネルの性能に対する影響を調査するため、以下のベンチマークを利用してオーバヘッドの計測を実施した。

(1) kernel build
http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.28.tar.bz2

(2) chat ベンチ
http://lbs.sourceforge.net/chat/chat-1.0.1.tar.gz

(3) dbench
ftp://ftp.samba.org/pub/tridge/dbench/dbench-3.03.tar.gz

(4) lmbench
http://www.bitmover.com/lmbench/

(5) tiobench
http://sourceforge.net/projects/tiobench/

(6) apache bench
http://httpd.apache.org/ (Apache 付属)

(7) ttcp
http://ftp.arl.mil/ftp/pub/ttcp/

それぞれのベンチマークにおいて、次の 5 パターンの設定を行って計測を実施した。

(1) pure
LKST の機能が組み込まれていないカーネルを利用して計測。

(2) LKST norec
LKST の機能が組み込まれたカーネルを利用し、全てのフックポイントを無効にした状態で計測。

(3) LKST IPA
LKST 性能評価機能で利用されるフックポイントのみを有効にした状態で計測。

(4) LKST nolock
LKST 性能評価機能で利用されるフックポイントのうち、スピンロック関連のフックポイントを無効にし、その他のフックポイントを有効にした状態で計測。

(5) LKST ALL
LKST で記録可能な全てのフックポイントを有効にした状態で計測。

それぞれのベンチマークは、各 5 回ずつ実行し、得られた結果の平均値を測定結果とした。

3.4.1. 評価環境

(1) システム環境

評価を行ったシステム環境を表 3.4-1 に示す。

表 3.4-1 評価を行ったシステム環境

項番	項目		評価環境
1	ハードウェア	CPU	Xeon 2.80GHz HT × 1 個
2		メモリ	1GByte
3		ハードディスク	36GB(U320 SCSI)
4	ソフトウェア	カーネル	MIRACLE LINUX V3.0 +新 LKST
5		解析ツール	lkstla
6		負荷ツール	カーネルビルド、chat ベンチ

3.4.2. ベンチマーク測定結果

各ベンチマークによる性能測定結果をグラフにまとめた。それぞれのグラフは、LKST 機能を含まないカーネルによる測定結果(pure)を基準値として 1 とし、その他の項目は pure に対する相対値で表している。

3.4.2.1. kernel ビルド

Linux カーネルのビルドは、多数のソースファイルをコンパイルすることより、CPU 性能、ディスク性能などのハードウェア的な要因と、スケジューリング、I/O 処理、プロセス管理、メモリ管理などのカーネルの実装に基づくソフトウェア的な要因に関連する様々な種類の処理が行われる。そのため、Linux 開発者の間では、カーネルのビルド時間をベンチマーク的な指標の 1 つとして用いることがある。

今回は Linux kernel 2.4.28 のビルドにかかる時間を計測した。

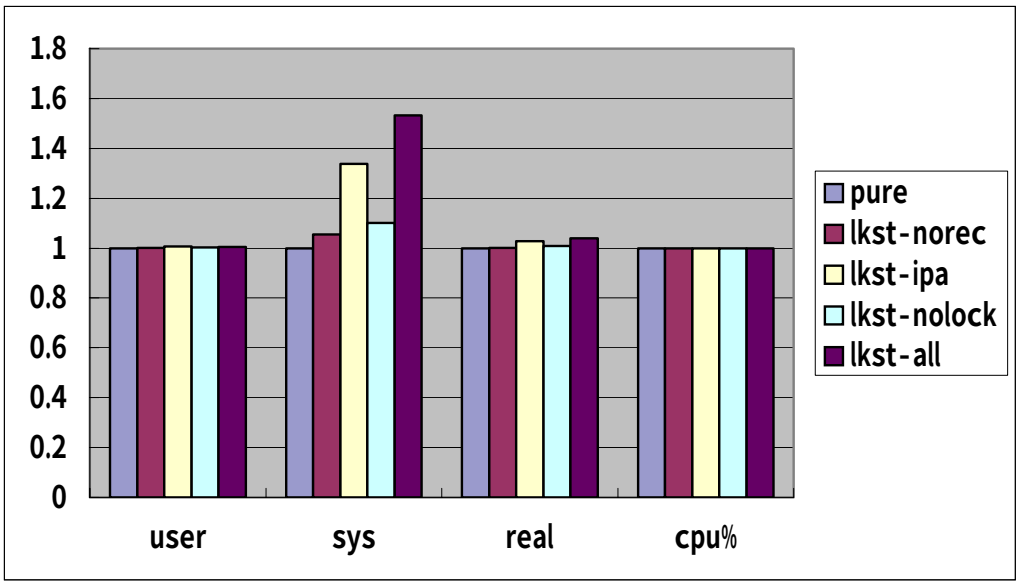

図 3.4-1 カーネルのビルド時間の計測結果の比較

図 3.4-1 は、カーネルのビルド時間にかかった user time、system time、real time、cpu 利用率の比較である。

LKST 性能評価機能が与える実際の影響の割合を確認するために、user time と system time の合計時間を図 3.4-2 に示す。

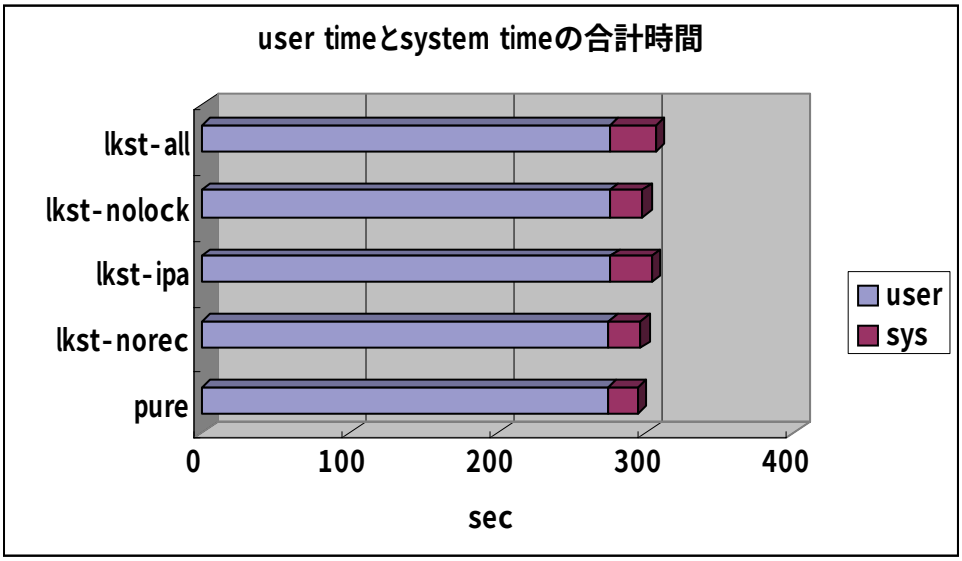

図 3.4-2 カーネルビルド時の user time と system time の合計時間

3.4.2.2. chat ベンチ

chat ベンチはサーバとクライアント間のネットワークを経由して、多数のメッセージ交換を行い、1 秒あたりにいくつのメッセージを交換することができるか計測するベンチマークである。

今回は同一サーバ内にサーバ、クライアントを起動しループバックデバイス(127.0.0.1)経由でメッセージ交換を行うベンチマークを実行した。

chat ベンチの計測結果を図 3.4-3 に示す。

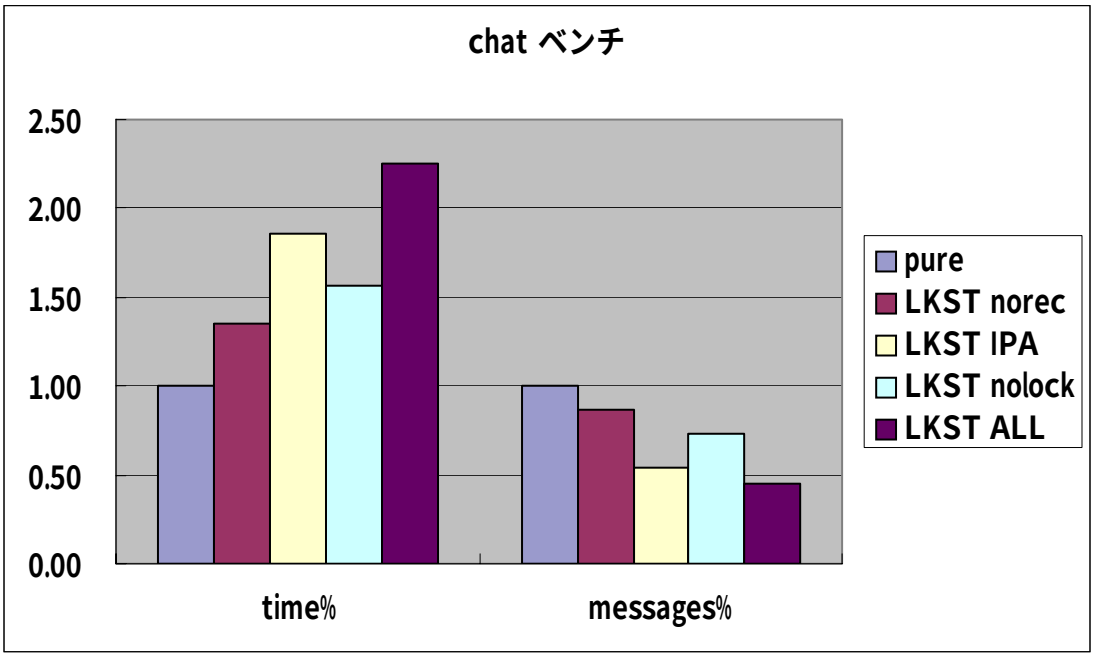

図 3.4-3 chat ベンチの測定結果の比較

図 3.4-3 の time%は chat ベンチの実行時間を表し、数値が小さいほど性能が高いことを表す。messages%は平均スループットを表し、数値が大きいほど性能が高いことを表す。

3.4.2.3. dbench

dbench は、Samba サーバが発行する I/O 要求を模してローカルファイルシステムのスループットを計測することができるベンチマークである。また、複数のクライアントからの同時接続を想定した負荷を発生させることができる。

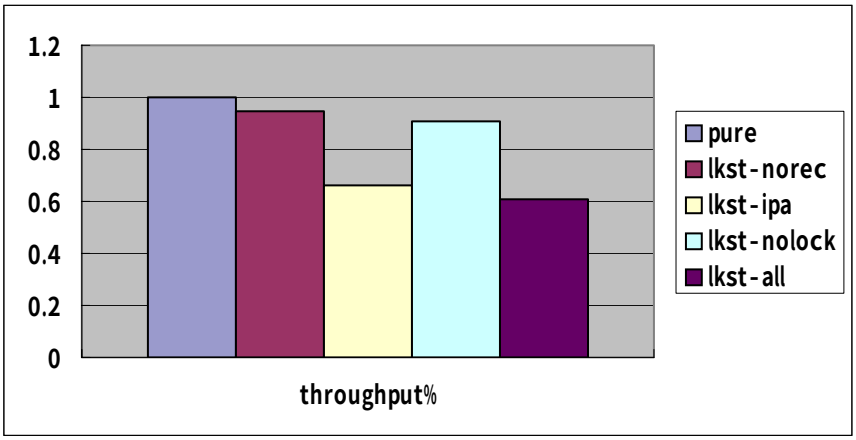

図 3.4-4 dbench の測定結果の比較

dbench によるベンチマークの計測結果を図 3.4-4 に示す。計測結果はスループットを表し、数値が大きいほど性能が高いことを表す。

3.4.2.4. LMbench

LMbench は、UNIX の基本機能の性能をはかるためのマイクロ・ベンチマークである。LMbench を利用することで、次のような機能のベンチマークを行うことができる。

(1) メモリ管理
(2) プロセス管理
(3) ネットワーク
(4) シグナル処理
(5) ファイル操作
(6) システムコール

lmbench-3.0-a4 を利用したベンチマークの測定結果を図 3.4-9 から図 3.4-9 に示す。

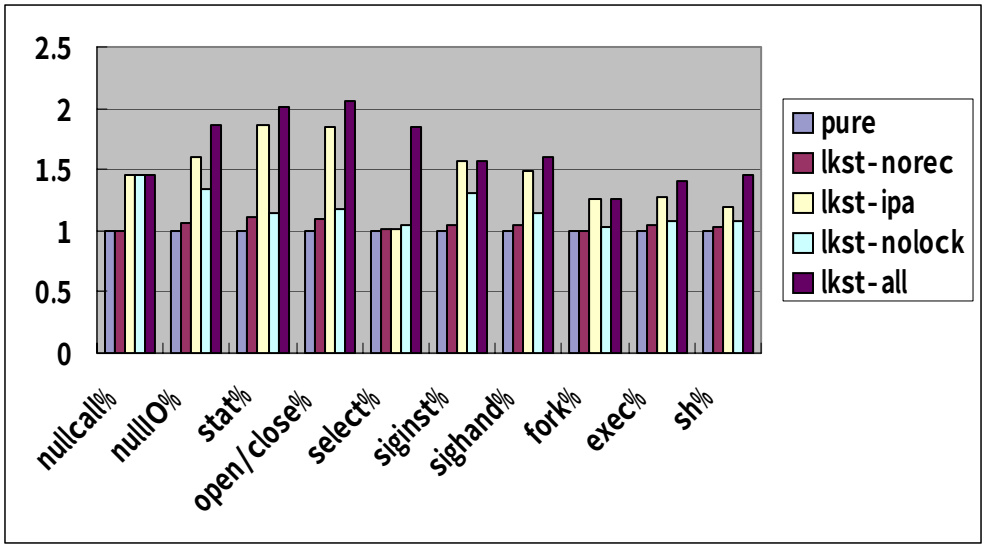

図 3.4-5 LMBench の測定結果(システムコール、シグナル等)の比較

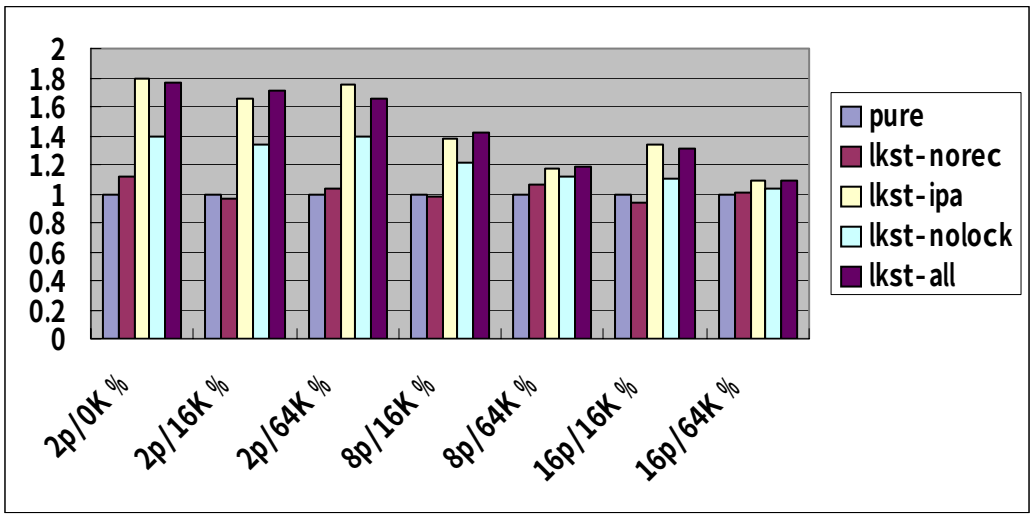

図 3.4-6 LMBench の測定結果(コンテキストスイッチ)の比較

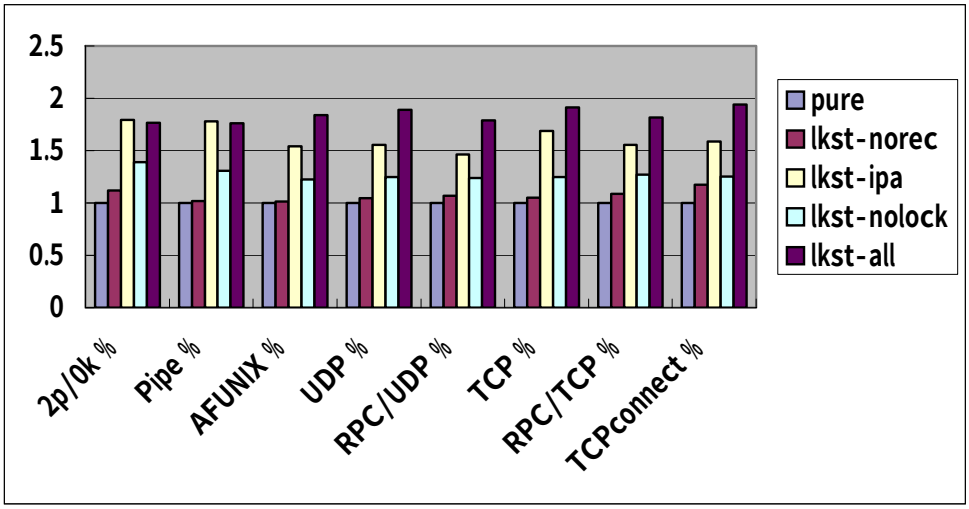

図 3.4-7 LMBench の測定結果(ネットワーク等)の比較

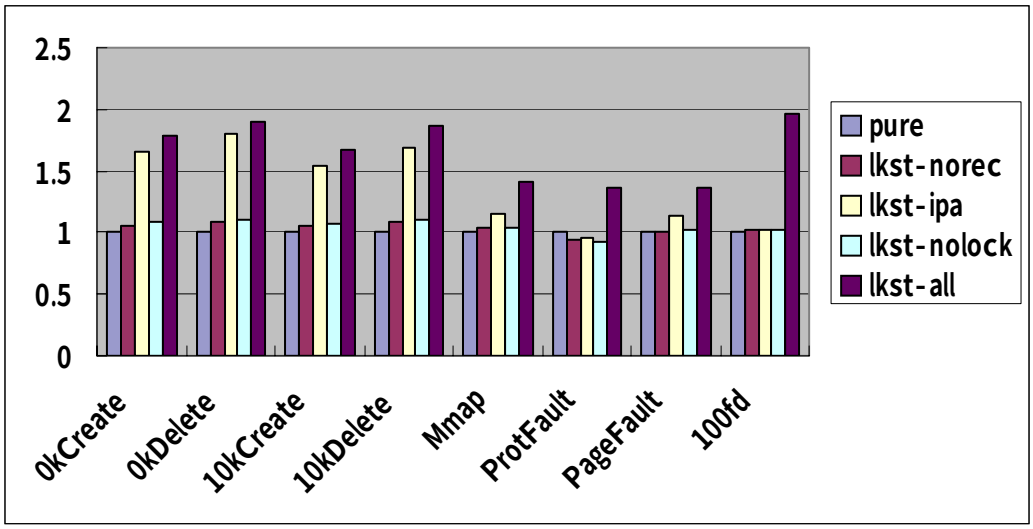

図 3.4-8 LMBench の測定結果(ファイル操作、VM 性能等)の比較

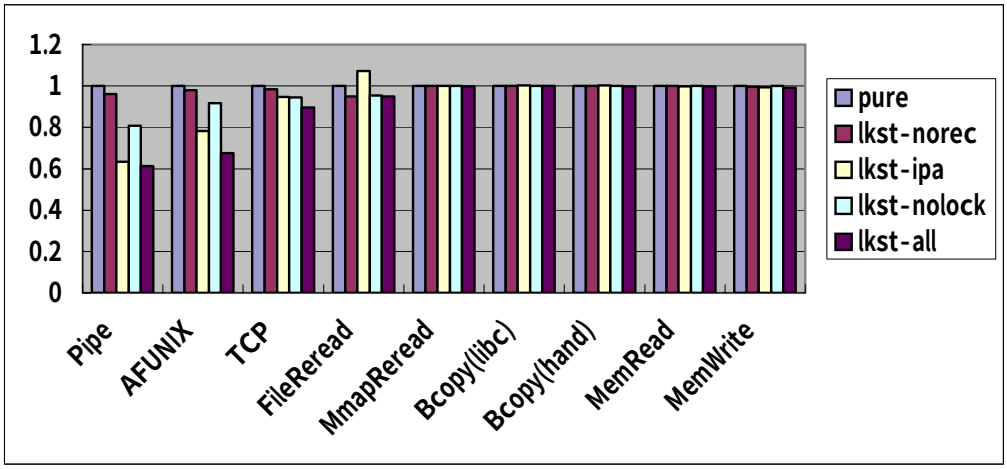

図 3.4-9 LMBench の測定結果(ローカルコミュニケーション)の比較

3.4.2.5. tiobench

tiobench は同時 I/O 性能を計測するためのベンチマークである。

Sequential Reads、Random Reads、Sequential Writes、Random Writes の 4 パターンの同時 I/O 性能を測定することができ、同時に I/O を発行するスレッドをパラメータによって増減することができる。

tiobench-0.3.3 を利用したベンチマークの測定結果を図 3.4-13 から図 3.4-13 に示す。

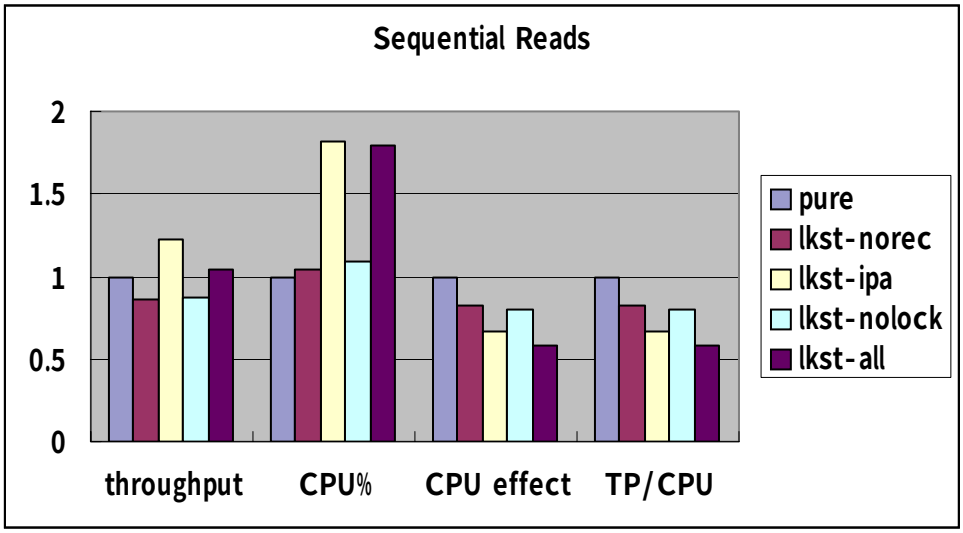

図 3.4-10 tiobench の Sequential Read の測定結果の比較

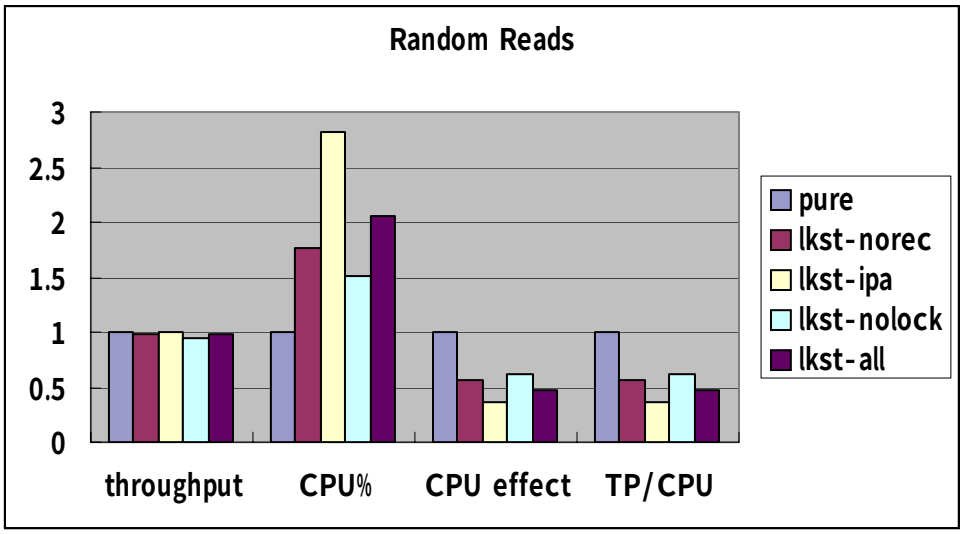

図 3.4-11 tiobench の Random Reads の測定結果の比較

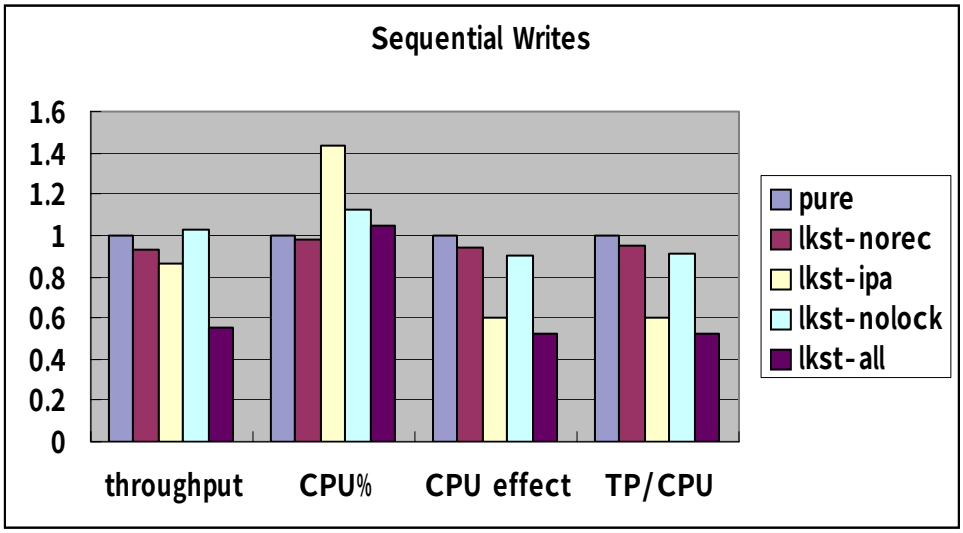

図 3.4-12 tiobench の Sequential Writes の測定結果の比較

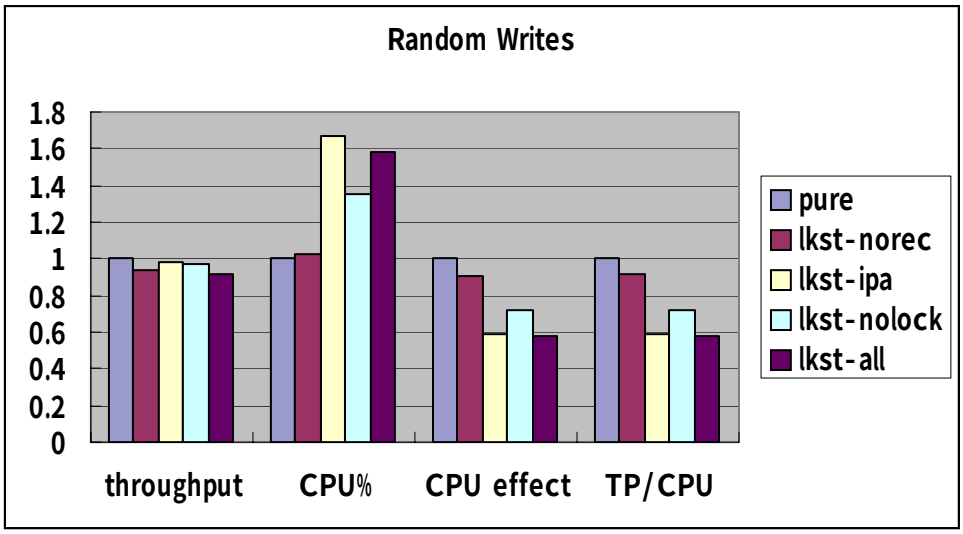

図 3.4-13 tiobench の Random Writes の測定結果の比較

ベンチマーク結果として計測されたスコアをまとめた結果を図 3.4-14 に示す。

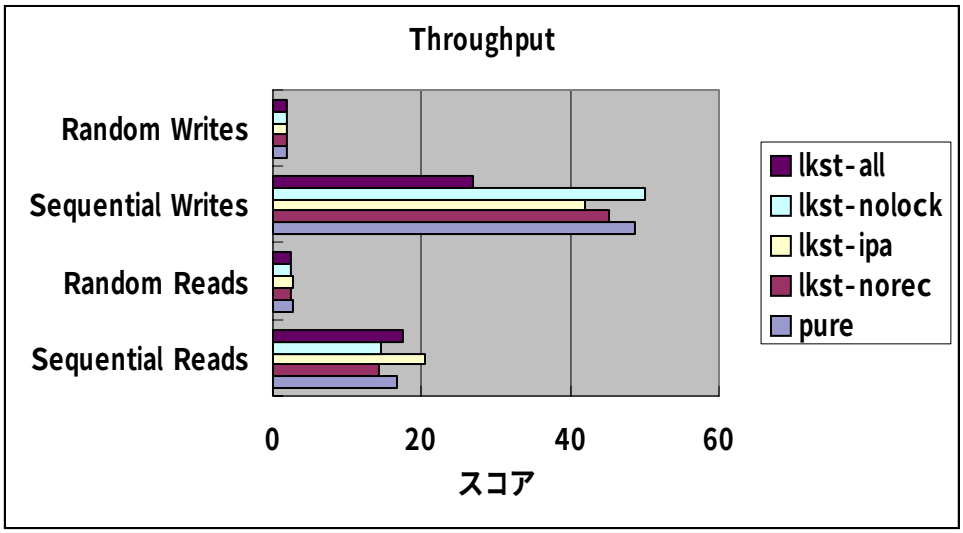

図 3.4-14 tiobench の測定結果(Throughput)

図 3.4-13、図 3.4-14 の計測結果の一部で、pure の結果を上回る結果が計測されているものがある。これは LKST のオーバヘッド以外に、計測結果を大きく左右する要因があるためと考えられるが、要因の特定には至らなかった。

3.4.2.6. apache bench

apache bench は、Apache に付属するベンチマークツールであり、Web サーバへのアクセスを繰り返すベンチマークである。

今回の検証では、同一構成の 2 台のサーバ間を Gigabit Ethernet で接続し、1 台をクライアント、1 台を Web サーバとしてベンチマークを実施した。

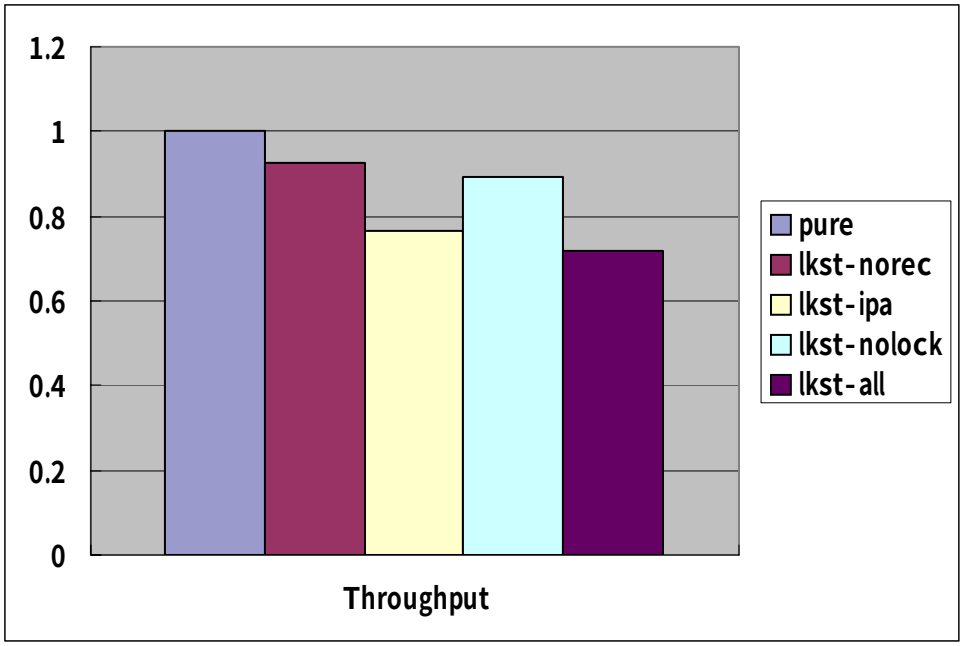

図 3.4-15 apache bench の測定結果の比較

図 3.4-15 は、apache bench によって得られたスループットの比較結果である。

3.4.2.7. ttcp

ttcp は 2 台のサーバ間をネットワーク経由でデータ転送し、ネットワークのバンド帯域を測定するベンチマークツールである。

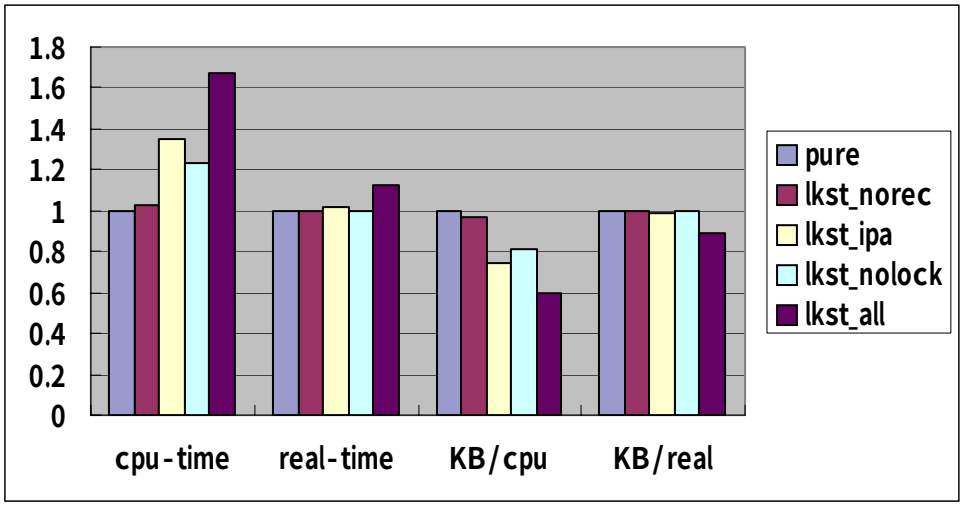

図 3.4-16 ttcp の測定結果の比較

図 3.4-17 に ttcp でデータの転送に要した時間をまとめた結果を示す。

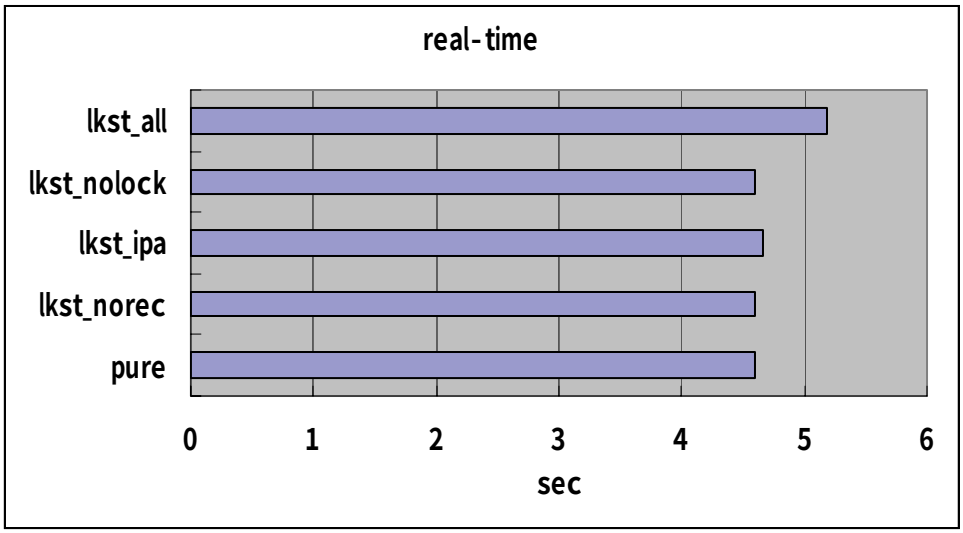

図 3.4-17 データの転送に要した時間(real time)

3.4.3. ベンチマーク結果のまとめ

ベンチマーク結果より、次のことがわかる。

(1) LKST 性能評価機能を組み込むことで、オーバヘッドが生じる。

(2) 記録するイベントの種類を増やすことによって、オーバヘッドは大きくなる。

(3) LKST を組み込むことでイベントを全く記録しない場合でもオーバヘッドが生じる

(4) ベンチマークの種類によって、オーバヘッドによる影響度が異なる

LKST を組み込むことによるオーバヘッドは、主にカーネル内部のさまざまな箇所に設けられたフックポイントにおいてイベントの記録を行うことによって発生する。従って、それぞれのベンチマークの実行過程において、いくつのフックポイントを通過するかによって生じるオーバヘッドは異なる。

今回実施したベンチマークは、kernel ビルドを除きマイクロ・ベンチマークと呼ばれる種類のベンチマークであった。これらのベンチマークは特定の処理やシステムコールを繰り返し、カーネル内部の処理を頻繁に繰り返す種類のベンチマークである。そのため、今回の計測では、フックポイントを頻繁に通過した結果、明確にオーバヘッドが計測されていると考えられる。

一方で、より実アプリケーションの実行環境に近い kernel ビルドによる性能測定では、system time のみ消費時間の増加が計測されているが、user time の消費時間は増加していない。これは、カーネル内部でイベントの記録を行う LKST の実装と照らし合わせても妥当な結果といえる。また、図 3.4-2 より system time の増加時間は、user time の実行時間と比較すると非常に小さいため、カーネルビルド全体の処理時間にはほとんど影響がでていない。

つまり、通常のアプリケーションの実行環境であれば、LKST 性能評価機能の組み込みによるオーバヘッドは今回の計測結果より少なくなると予想できる。

また、今回は性能測定の際、一度に様々なデータを取得するために複数のフックポイントを有効にしたが、取得が必要なデータを絞り込み、有効にするフックポイントの数を減らすことで、オーバヘッドをより削減することも可能である。

3.4.4. 測定データ

参考のために、今回測定したベンチマークの測定データを掲載する。

3.4.4.1. kernel ビルド

表 3.4-2 Kernel ビルドの測定データ

	user	sys	real	cpu%
pure	274.41	20.34	02:29.65	197
LKST norec	274.89	21.46	02:29.87	197
LKST IPA	276.47	27.23	02:33.71	197
LKST nolock	275.35	22.38	02:31.02	197
LKST ALL	275.56	31.18	02:35.69	197

3.4.4.2. chat ベンチ

表 3.4-3 chat ベンチの測定データ

	time	messages
pure	2.2905	391509.17
LKST norec	3.0915	338881.83
LKST IPA	4.2535	210165.50
LKST nolock	3.5963	286245.00
LKST ALL	5.1650	175315.50

3.4.4.3. dbench

表 3.4-4 dbench の測定データ

	throughput
pure	173.55
lkst-norec	164.40
lkst-ipa	115.23
lkst-nolock	158.03
lkst-all	105.20

3.4.4.4. LMbench

表 3.4-5 lmbench の測定データ(システムコール、シグナル等)

	nullcall	nullIO	stat	open close	select	siginst	sighand	fork	exec	sh
pure	0.46	0.69	3.65	4.98	19.83	0.75	2.50	265.00	795.50	2771.50
lkst-norec	0.46	0.73	4.03	5.48	19.93	0.79	2.63	264.25	833.50	2868.25
lkst-ipa	0.67	1.11	6.79	9.17	20.13	1.18	3.73	331.50	1013.50	3316.50
lkst-nolock	0.67	0.92	4.19	5.82	20.78	0.98	2.86	273.00	858.75	2977.75
lkst-all	0.67	1.28	7.33	10.23	36.50	1.18	4.00	333.75	1122	4041.75

表 3.4-6 lmbench の測定データ(コンテキストスイッチ)

	2p/0K	2p/16K	2p/64K	8p/16K	8p/64K	16p/16K	16p/64K
pure	1.78	1.93	2.05	3.64	22.95	7.01	31.35
lkst-norec	1.99	1.87	2.11	3.58	24.30	6.56	31.38
lkst-ipa	3.19	3.20	3.60	5.00	27.03	9.37	34.13
lkst-nolock	2.47	2.57	2.85	4.42	25.50	7.75	32.55
lkst-all	3.14	3.29	3.40	5.16	27.33	9.23	33.98

表 3.4-7 lmbench の測定データ(ネットワーク等)

	2p/0k	Pipe	AFUNIX	UDP	RPC/UDP	TCP	RPC/TCP	TCPconnect
pure	1.78	7.87	14.95	15.80	25.28	18.70	30.45	48.50
lkst-norec	1.99	8.02	15.15	16.53	27.05	19.68	33.08	57.00
lkst-ipa	3.19	14.00	23.05	24.55	37.05	31.55	47.35	77.00
lkst-nolock	2.47	10.30	18.33	19.70	31.28	23.30	38.70	60.75
lkst-all	3.14	13.88	27.53	29.85	45.20	35.73	55.33	94.00

表 3.4-8 lmbench の測定データ(ファイル操作、VM 性能等)

	0kCreate	0kDelete	10kCreate	10kDelete	Mmap	ProtFault	PageFault	100fd
pure	27.23	24.33	100.68	44.55	1305.00	0.88	3.29	18.28
lkst-norec	28.93	26.45	106.13	48.53	1355.75	0.83	3.33	18.63
lkst-ipa	45.20	43.80	155.75	75.15	1513.75	0.85	3.72	18.83
lkst-nolock	29.78	26.85	108.60	49.10	1345.50	0.81	3.35	18.7
lkst-all	48.43	46.35	167.75	83.10	1834.00	1.20	4.46	35.9

表 3.4-9 lmbench の測定データ(ローカルコミュニケーション)

	Pipe	AFUNIX	TCP	FileReread	MmapReread	Bcopy(libc)	Bcopy(hand)	MemRead	MemWrite
pure	658.50	2900.25	379.25	1528.90	1997.73	673.73	694.90	1990.50	940.35
lkst-norec	632.75	2841.25	373.50	1452.60	1998.03	673.23	695.48	1989.75	936.25
lkst-ipa	416.75	2270.50	359.50	1640.60	1997.30	675.33	696.50	1988.50	934.18
lkst-nolock	531.75	2657.50	358.00	1458.83	1997.90	673.98	694.90	1990.00	940.88
lkst-all	404.00	1957.50	339.50	1450.53	1995.78	673.55	693.73	1987.50	932.13

3.4.4.5. tiobench

表 3.4-10 tiobench の測定データ(Sequential Read)

Sequential Reads	throughput	CPU%	avg latency	max latency	lat >2s %	lat >10s %	CPU effect	TP/CPU
pure	16.65	6%	1.870	463.91	0	0	281	280.586451
lkst-norec	14.29	6%	2.179	327.91	0	0	232	231.642081
lkst-ipa	20.32	11%	1.480	425.95	0	0	189	188.672238
lkst-nolock	14.52	6%	2.143	468.90	0	0	225	224.559233
lkst-all	17.36	10.62%	8.341	1933.29	0	0	163	163.465160

表 3.4-11 tiobench の測定データ(Random Reads)

Random Reads	throughput	CPU%	avg latency	max latency	lat >2s %	lat >10s %	CPU effect	TP/CPU
pure	2.56	1%	11.517	77.75	0	0	312	312.195122
lkst-norec	2.52	1%	11.711	64.39	0	0	174	173.673329
lkst-ipa	2.57	2%	11.602	65.08	0	0	112	111.545139
lkst-nolock	2.43	1%	12.171	70	0	0	195	195.494771
lkst-all	2.52	1.68%	56.981	331.62	0	0	150	150

表 3.4-12 tiobench の測定データ(Sequential Writes)

Sequential Writes	throughput	CPU%	avg latency	max latency	lat >2s %	lat >10s %	CPU effect	TP/CPU
pure	48.72	67%	0.454	1835.05	0	0	73	72.7381308
lkst-norec	45.12	65%	0.549	2303.34	0.00381	0	69	68.9065363
lkst-ipa	42.01	96%	0.468	1649.06	0	0	44	43.7649755
lkst-nolock	50.06	75%	0.469	1681.64	0	0	66	66.3661673
lkst-all	26.84	69.97%	4.304	35331.98	0.00877	0.00724	38	38.3592968

表 3.4-13 tiobench の測定データ(Random Writes)

Random Writes	throughput	CPU%	avg latency	max latency	lat >2s %	lat >10s %	CPU effect	TP/CPU
pure	1.93	1%	0.057	37.31	0	0	156	155.896607
lkst-norec	1.81	1%	0.028	5.04	0	0	142	142.072214
lkst-ipa	1.9	2%	0.404	1407.52	0	0	92	91.6988417
lkst-nolock	1.87	2%	0.029	5.09	0	0	112	111.575179
lkst-all	1.76	1.96%	0.688	1974.42	0	0	90	89.841756

3.4.4.6. apache bench

表 3.4-14 apache bench の測定データ

	Throughput
pure	8088.822
lkst-norec	7480.770
lkst-ipa	6176.808
lkst-nolock	7226.596
lkst-all	5808.918

3.4.4.7. ttcp

表 3.4-15 ttcp の測定データ

	cpu-time	real-time	KB/cpu	KB/real
pure	1.804	4.601482	291627.6	113939.00
lkst_norec	1.858	4.600456	282684.6	113965.00
lkst_ipa	2.440	4.667174	215402.8	112335.80
lkst_nolock	2.222	4.601430	236793.0	113940.20
lkst_all	3.026	5.186406	173965.8	101116.76

4. 総括

4.1. 性能評価ツール(lkstlogtools)の有効性

(1) lkstlogtools により、1.3 で課題として記載した以下のことを実現できた。
- 性能評価を行なうための情報の収集・評価手段の実現
- 表 1.3-1 に挙げた性能評価用情報の収集および解析ツールの実現

(2) これまで sar によって、最小 1 秒間隔程度でしか取得できなかったカーネル内部の詳細な情報を、lkstlogtools により正確に記録することが可能になった。

(3) ３章の考察からもわかるように、lkstlogtools は、性能の分析、特に処理時間の分布と遅延の分析をより詳細に行なうことができる。視覚的に認識できるため、解析も行ないやすい。

(4) lkstlogtools は、性能遅延を見つけた時に、その後に行なう原因究明に対して、調査すべき時間帯、関連するプロセスなどを絞り込むことに有効に活用できる。また、その情報をキーに関連する別のデータについて分析を行い、複数の解析結果を照らし合わせて多面的な解析を行うことが出来る。

(5) lkstlogtools の中の解析ツール lkstla は、解析ルーチンの追加が行なえるように開発している。このため、解析中に取得している別のデータの検出や、別の評価計算を行ないたい場合、それらへの対応を容易に行なうことができた。

(6) 従来の LKST はイベントの記録のみの機能であり、その解析もテキストのログのみであったため、利用に対する敷居が非常に高かった。一方、今回開発した lkstlogtools は、障害解析、パフォーマンスチューニングなど様々な利用方法が想定され、さらに可視化ツールもあわせて提供することによって、カーネルに関する知識が十分でなくとも、システムの状態を把握するための情報を簡単に取得できるようになった。

4.2. 性能評価ツールの開発規模

今回の lkstlogtools の開発規模は、ファイル数 93、合計 6636 steps である。このうちカーネルへのパッチは 8 パッチ、合計 332steps であり、LKST に追加・修正した性能測定用のイベント数は 36 イベントである。

また、MIRACLE LINUX V3.0 への移植による開発規模は、LKST 2.2.1 に対する kernel 2.4 対応のパッチ行数が 791steps、lkstlogtools に対して kernel2.4 対応のための MIRACLE LINUX のカーネルへのパッチ行数は 393steps、解析ツールへのパッチ行数は 20steps である。また、RPM パッケージの修正は 91steps である。

4.3. 今後の課題

4.3.1. LKST 本体の課題

(1) バッファ消費量の指標：
LKST で情報を記録する際、イベントの発生頻度とバッファのサイズによっては欲しい情報が上書きで消されてしまうことがあった。記録するイベントの数や種類と、それらの記録頻度及びバッファの消費速度に関しては、なんらかの指標を提供する必要があると考える。

(2) 適応型マスクセット：
イベントの発生数が多くなるとトレースのオーバヘッドが大きくなり、OS の処理に影響を与えてしまう。トレース対象イベントを動的に切替える機能を利用し、負荷に応じて自動でマスクセットを変更するなどの仕組みでオーバヘッドの軽減を図る手段が必要と考える。

(3) 読み出しオーバヘッド低減：
イベントの情報をユーザプログラムに渡す部分で、何らかの方法でメモリコピーのオーバヘッドを減らす必要性があると考える。また、記録した情報をディスクに保存する場合、ディスク IO が頻発している状態であると、LKST 自身の記録が待たされてしまう。これに対応するため、ネットワークを使った転送や、イベントの圧縮、あるいはカーネルデーモンを使ったディスクへの保存などが考えられる。

(4) 記録時のオーバヘッド低減：
LKST を組み込むことで、イベントを全く記録しない場合でも数%程度の性能劣化が発生することが確認できた。今後の普及を図るためには、LKST の機能を利用しないときには、たとえ数%であっても性能劣化の発生を抑止することが望まれる。

(5) サンプリング機能：
LKST の取得するイベントによっては、詳細にすべてを記録するよりも、いくつかのサンプリングを取ったほうがデータ数を抑制しやすいものがある。イベントハンドラを拡張するか、フックポイントを改造するなどして、特定の場所のイベントはサンプリングするようにしてデータ数を抑制することが望まれる。

4.3.2. 性能評価ツールの課題

(1) GUI ツールの開発：
個別の性能についてのデータ解析は可能であるが、複数の解析結果の比較のための機能が少ない。また、ある解析結果から、別のデータを解析するための絞込み条件（PID や時間など）を作るのに手間がかかる。複数の解析結果を同時に表示可能な、インタラクティブな GUI の可視化ツールがあれば、比較を高速に出来るのではないかと考える。

(2) イベントフィルタの拡充：
現在の解析ツールは、必要最小限のイベントフィルタリングの機能をサポートしている。しかし解析に使ってみた結果、ある閾値を超えた場合や、出力パターンフィルタなどの複雑な絞り込みも有用であることがわかった。イベントフィルタリングの機能を拡充する必要があると考える。

(3) 中間データファイル生成による効率化：
現在の解析ツールは、解析結果の表示形式やフィルタを変えたいだけの場合でも、再度データの解析を実行するようになっている。データファイルが大きくなると、一度の解析に時間がかかるため、解析データの絞込みを繰り返し行うことが難しい。解析だけを行った中間データファイルから、フィルタリングや表示形式の変換が出来るようにすることで、解析の効率化が図れると考える。

(4) アラーム機能：
現在の解析ツールは、アプリケーション実行後にデータを取得し、解析を行っているが、いくつかの解析ルーチンについては、LKST の拡張イベントハンドラとしてカーネル内部で実行することが可能である。こうすることで、アプリケーション実行中に解析を行い、何らかの警告ログを出すなどの仕組みは実現できる可能性が出てきた。本機能の必要性が高ければ、オーバヘッドとのトレードオフを含めて検討してみたい。

最後に、従来、今回の開発成果、将来的な目標について、機能面での比較を行ったものを表 4.3-1 に示す。

表 4.3-1　機能面での比較

	従来	今回の開発成果	将来的な目標
機能	LKST	LKST + lkstlogtools	強化版 LKST + 強化版 lkstlogtools + GUI
カーネルの内部情報の記録	◎	◎	◎
性能評価情報の記録	×	◎	◎
記録した情報の解析	△	○	◎
複合的な情報解析	×	△	◎
情報の可視化	×	○	◎

◎：十分に行うことが出来る
○：一部を手動で行う必要がある
△：手動で行う必要がある
×：出来ない

本書は、独立行政法人 情報処理推進機構から以下の８社への委託開発の成果として作成されたものです。
委託先企業：（株）日立製作所（幹事会社）
（株）SRA、（株）NTT データ、新日鉄ソリューションズ（株）、
住商情報システム（株）、（株）野村総合研究所、ミラクル・リナックス（株）、
ユニアデックス（株）　（五十音順）