Efficiently Scheduling X Clients

Keith Packard
XFree86 Core Team, SuSE Inc.
keithp@suse.com

Abstract

The X server is charged with providing window system services to many applications simultaneously, and needs a scheduling mechanism to distribute it's limited resources among these applications. The original scheduling mechanism was simplistic and caused graphics-intensive applications to starve interactive applications.

A new scheduling mechanism has been designed which fairly distributes time among the requesting applications while at the same time increasing performance by a small amount. Descriptions of the original and new scheduling mechanism and empirical measurements demonstrating the effects of scheduling within the X server are included along with a discussion on how the design was arrived at.

1 Problems to be solved

1.1 User Feedback Latency

A window system provides an interface between applications and the user, it doesn't receive, transmit or process the underlying data. As such, it focuses on collecting user input, distributing it among the applications and displaying the resulting changes in application state to the screen. For a satisfactory user experience, the delay between event generation and the change in display should be small. As the window system is dependent on the application to generate the appropriate display changes, the delays within the window system should be kept as small as practical.

1.2 Smooth Animation

While the eye will accept more jitter than the ear, animations still require relatively consistent delays between frames. For the window system, this means that when time is scarce, it must parcel it fairly among applications and with fine enough granularity to provide a smooth degradation in frame rates.

1.3 Inter-client synchronization

The X Sync Extension [GCGW91] provides a way of synchronizing X clients with other X clients and various other events. It also provides priorities which are used to schedule among active clients. For this to work well, the X server must minimize the time needed to recognize clients when they become active.

1.4 Speed

Advances in hardware have made the core X graphics operations fast enough in most environments. However, changes which reduce performance are never well received. While X provides for simultaneous active windows, frequently only a single application is drawing. Especially when measuring performance with benchmarks. Any improvements in behavior under heavy load must not negatively impact the common case of a single busy application.

2 Characterizing Scheduler Performance

To effectively analyze the performance of scheduling changes some empirical measurements are helpful.

Based on the problems identified above, the following measurements can be made:

2.1 Measuring Latency

Each X event contains a timestamp marking when the event was generated within the X server. For input events, this should mark the time of the physical action. However, the Linux kernel doesn't provide timestamps marking when the event was received by the kernel, so the X server marks the time they were received by the X server. For mouse events, this is done from within a SIGIO handler and so is relatively close to the physical time.

Applications can cause the X server to generate timestamped PropertyNotify events. By doing this after rendering the feedback and then measuring the difference between the event time and the associated property notify time, a reasonable measurement of user latency can be obtained.

2.2 Animation Update Jitter

Because the X protocol is asynchronous, there can be a large delay from when the application makes a rendering request to when the resulting update appears on the screen. While this delay is important, the jitter generated by differences in scheduling are more visible to the user. Therefore, two different measurements are needed to accurately assess the performance of the system. The delay between when the application generates the rendering requests and when they are displayed measures the latency. The time between displayed frames provides a measure of the jitter in the display.

Again, PropertyNotify events can be used to determine the time at which rendering was completed. The X server computes timestamps for events using the system clock, so as long as the application and X server share the same clock, the delay from generation to display can be measured.

2.3 Measuring Speed

X11perf [MKA+94] is an application used to analyze X server performance and to generate benchmark numbers. This tool can be used to accurately measure the performance impact of changes made to the X server.

3 Existing Practice

The original scheduling algorithm in the X server as developed at Digital in 1987 was relatively simplistic. X applications were modest in their demands on the server as the bulk were simple text-based applications such as emacs or xterm.

Soon thereafter some simple demonstration applications were written which would render the X server essentially unusable. Plaid is one of these. It sends an endless stream of rendering requests to the server of just the right length to tie up the system for long periods of time.

Real applications with such behavior were not far behind, today users have a wide variety of increasingly complex applications that ask much from our venerable protocol. The primitive scheduling which was an occasional annoyance ten years ago has become more of a problem today.

3.1 The Original Algorithm

The sample X server is a single-threaded network server, each client connects to the service using a well known port and sends requests over that connection for processing by the server. Replies, asynchronous events and errors are returned over the same link. As with many single-threaded network server applications, the X server uses `select(2)` to detect clients with pending input.

Once the set of clients with pending input is determined, the X server starts executing requests from the client with the smallest file descriptor. Each client has a buffer which is used to read some data from the network connection, that buffer can be resized to hold unusually large requests, but is typically 4KB. Requests are executed from each client until either the buffer is exhausted of complete requests or after ten requests.

After requests are read from all of the ready clients, the server determines whether any clients still have complete requests in their buffers. If so, the server foregoes the `select(2)` call and goes back to processing requests for those clients. When all client input buffers are exhausted of complete requests, the X server returns to `select(2)` to await additional data.

3.2 Analysis of this Algorithm

One problem with this algorithm is that it uses a poor metric for the amount of work done by applications. In most cases, ten requests can be executed very rapidly. Applications tend to render only a few objects in each request and the rendering requests are typically quite simple. It is possible, however, to generate requests which run for quite some time, X requests can contain thousands of primitive objects. Ten such requests could take several hundreds of milliseconds to execute. This leads to large variations in the amount of time devoted to each application.

Another issue is that applications which generate few requests are starved for attention. As the X server busily empties the buffers from more active clients, it remains deaf to those with more modest demands. Until every client request buffer is empty of requests, the server doesn't check on the remaining clients.

Less obviously, the server stops processing requests for a client whenever the request buffer doesn't contain a complete request. This means that for rapidly executed large requests, the server calls `select(2)` more often as the request buffer will hold fewer requests.

The benefit of this system is that when clients are busy, the server spends most of its time executing requests and wastes little time on system calls.

4 Available Scheduling Parameters

The information available and the execution context control the design of a scheduler as much as the desired performance characteristics. The X server is a user-mode process written in a high-level language and is designed to be relatively independent of operating system and hardware architecture. This makes the information available limited to that provided by common OS interfaces and library functions.

4.1 Request Atomicity

The X protocol requires that

> the overall effect must be as if individual requests are executed to completion in some serial order, and requests from a given connection must be executed in delivery order (that is, the total execution order is a shuffle of the individual streams). [SG92]

This makes suspending request processing in the middle of a request difficult (in general) as objects manipulated by each request must not appear to other clients in any intermediate state. As a result, the design of the single threaded X server is largely predicated on scheduling clients at request boundaries. There are a few exceptions dealing with network font access which preserve atomicity by changing no global state before suspending request execution.

For the most part, this limitation is not too severe, X requests are usually small and execute quickly. However, when drawing lines and arcs thicker than a single pixel, the server can spend quite some time on a single request. Drawing arcs requires finding three solutions to an eighth order polynomial per scan-line and approximating an elliptical integral for dash lengths. Fortunately, applications largely avoid these primitives as they are so slow making them less of an issue.

Changing this architectural constraint is beyond the scope of this project and so the atomicity requirement translates into a scheduling granularity no finer than a request boundary.

4.2 System Performance

The cost of system calls and other operating-system procedures is important in determining how often each call can be made without a significant impact on overall performance.

Procedure	Time (μ seconds)
Null procedure activation	0.0470
Null syscall (getppid)	0.4708
Gettimeofday	0.8333
Read (1 byte) from /dev/zero	0.9789
Write (1 byte) to /dev/null	0.7709
Select on 10 fd's	9.3765
Signal handler overhead	3.158
UDP latency using localhost	77.3502
TCP latency using localhost	128.1683

Table 1: Performance for Linux System Operations

Measurements made with lmbench [MS96] on a Pentium 300 MHz Mobile MMX laptop by Theodore Ts'o yield the numbers shown in Table 1.

4.3 System Time

In any system which interacts with the user, one reasonable scheduling metric is real time. The X server would check the current time after each request and terminate processing for a client at the appropriate time. However, the current time is only available to user applications through a system call. The X server can execute an "NoOperation" request in 0.360 μ seconds, making it more than twice as fast as a call to `gettimeofday(2)`. Calling `gettimeofday(2)` after processing each request would be too expensive, so the X server needs a more efficient mechanism for obtaining the current time.

A low resolution clock can be generated using the interval timer mechanism (`setitimer(2)`). A measure of time can be computed by incrementing a global variable from the signal handler. Signals may occasionally be lost under heavy system load. Under heavy load, the X server would already be starved for CPU resources making scheduling decisions less precise in any case.

The X server should be idle when no work is to be done. When the `ITIMER_REAL` timer is used, `SIGALRM` is delivered even when the server is waiting. As the server suspends using `select(2)`, that system call will immediately return `EINTR` causing the server to execute a significant amount of code before re-entering `select(2)`. To eliminate the load of both a signal handler and another call to `select(2)` on the system, the X server temporarily disables the timer if it receives a signal while the server is suspended within `select(2)`. When the server awakens again, it restarts the timer.

The X server could use `ITIMER_VIRTUAL` which would report CPU time consumed by the X server. While the X server was idle, no signals would be generated. However, on a busy system, the X server would use progressively larger timer intervals. Additionally, the Linux kernel only accrues CPU time to a process when it happens to be running during the timer interrupt, occasionally increasing the scheduling interval.

4.4 User Intent

Unlike other window systems, X applications register the set of events they are interested in receiving so that unwanted events are not transmitted over the network. When an application receives a requested mouse or keyboard event, the server can infer that the application is likely to generate screen updates based on that event. The scheduler can grant additional resources to those applications.

The X server already used this to a limited extent. Events generated by the user are given priority and are delivered as soon as any current request completes. However, as mouse motion events can be numerous, they were not handled in this manner. Typical modern mouse devices generate fewer than 100 events per second when in motion, not a considerable burden for a modern machine. This behavior was changed to produce the results shown here. Without such changes, pointer motion events would be delayed until a normal scheduling interval. This caused long lags between pointer event generation and application receipt of that event.

4.5 Queued Requests

Another factor available to the scheduler is whether request buffers are empty, contain a partial request or contain one or more complete requests. The existing scheduler uses this to avoid making additional `select(2)` calls. The existence of a partial request makes it likely that the remaining data are waiting in an OS buffer and that a `read(2)` would likely result in additional data.

5 Proposed Scheduler

With the parameters outlined above, the design of the scheduler is straightforward. The goal is to provide relatively fine grained time-based scheduling with increased priority given to applications which receive user input.

5.1 Dynamic Priorities

Each client is assigned a base priority when it connects to the server. Requests are executed from the client with highest priority. Various events cause changes in priority as described below. The Sync extension also applies priorities, the scheduler priorities sort among clients with the same Sync priority so that applications can override the internally generated priorities.

Clients with the same priority are served in round-robin fashion, this keeps even many busy clients all making progress.

5.2 Time Based

Using the `ITIMER_REAL` mechanism outlined above, a copy of the system clock is kept in the X server address space, updated periodically while the X server is active. The resolution of the clock limits the granularity of scheduling and is set to 20ms.

Each client is allowed to execute requests for the same interval after which time other clients are given a turn. If the client is still running at the end of the interval, the client's priority is reduced by one (but no less than a minimum value). If the client hasn't been ready to run for some time, priority is increased by one (but no more than the initial base value). These priority adjustments penalize busy applications and praise idle ones. This is a simplification of discovering precisely how much time a client has used; that would require a system call.

Each time a client has finished running, the X server recomputes the set of clients ready for execution. That includes examining each client request buffer to determine whether a complete request is already available and also making a call to `select(2)` to discover whether any idle clients have delivered new requests. This is necessary to ensure that requests from higher priority clients are served within a reasonable interval independent of the number of busy low priority clients.

5.3 Monitor User Events

When a client receives user mouse or keyboard events, their priority is raised by one, but no more than a maximum value which is above the base priority. Just as in the Unix scheduler [Tho78], it is desirable to have applications which wait for user input to receive preferential treatment. However, there is no easy way to know whether the application is waiting for input in this case so instead we give modest praise when events are delivered.

5.4 Keep Reading From Clients

When a request buffer no longer contained a complete request, the original scheduler would stop processing a client and wait until `select(2)` indicated that additional data were available. Now `read(2)` is tried first. When that fails to fill the request buffer with a complete request, the server stops processing requests for that client. This increases the number of requests executed before another call to `select(2)` is made.

5.5 Lengthen Timeslice for Single Client

The scheduler monitors how many applications are making requests, when only a single client is making requests for an extended time (one second) the scheduler increases the amount of time allotted to that client. This improves performance for a single busy application. When another client makes a request, the timeslice interval is returned to normal. While this reduces interactive performance, the effect is transient as it is eliminated as soon as another application is ready to execute.

6 Experimental Setup

The machine used was a Compaq Prosignia with a 466MHz Celeron processor and an S3 Savage4 PCI video board with 32Meg of video ram.

In order to generate reasonable intervals in the animation test, the Linux 2.2.10 kernel was compiled with HZ set to 1000 instead of the default 100. This causes the hardware clock to be programmed for an interval of 1ms instead of the usual 10ms. An alternative would have been to increase the animation interval by a factor of 10, but that would have used an unrealistically low frame rate.

Because the measurements used timestamps reported in X events, they were only accurate to the nearest 1ms. This means that the measurement accuracy of shorter intervals is somewhat limited.

7 Results

To measure the effects of this scheduler, two test cases were written. They were first run on an otherwise idle X server, next they were run while twelve copies of an actively drawing application (plaid) were also running. The idle measurements were (as expected) identical between the two schedulers, only one copy of those results are included in the tables below.

7.1 Interactive Application

The first is a simple interactive feedback demonstration, a rubber band line is drawn following the cursor position. The delay between event generation and receipt is measured (Receipt), along with the total delay from event to the drawing of the line (Echo). See Figure 1.

The old scheduler suffers from significant latency both in delivering events and in scheduling the interactive client for execution. Subjectively, the feedback was jerky and lagged the actual pointer position by long intervals making it difficult to control the application.

The new scheduler also suffers additional latency when other applications are running, but total feedback delay is over 20 times lower (7.4 vs 170) when compared to the old scheduler. There was little perceptible difference between this and the Idle case. Feedback between the mouse and the display was slightly jerky but still quite usable.

7.2 Animation Application

The second test is an animation example. The “xengine” application was modified to delay 15ms between frames. The time between drawing a frame and the subsequent display on the screen was measured (Drawing), along with the time between frames (Frame). See Figure 2.

In this test, the old scheduler suffered from two problems, the long delay between image generation and display (219ms mean), and the large deviation in inter-frame intervals. The long delays would make synchronization with other media, such as audio, difficult. The large range of inter-frame intervals indicates that while busy with other clients, requests from the animation would queue up to be processed several frames at a time. The visible effect was a jerky sequence of images lacking a fluid sense of motion.

Visually, the new scheduler was difficult to distinguish from the idle case. However, the long timeslices granted to each client had measurable effects. The average delay between the generation of the frame and subsequent display is very close to half of the time slice interval. The deviation in frame intervals shows the same effect, that the scheduler was limited in precision by the slice interval.

With both of these tests, the new scheduler is measurably better than the old.

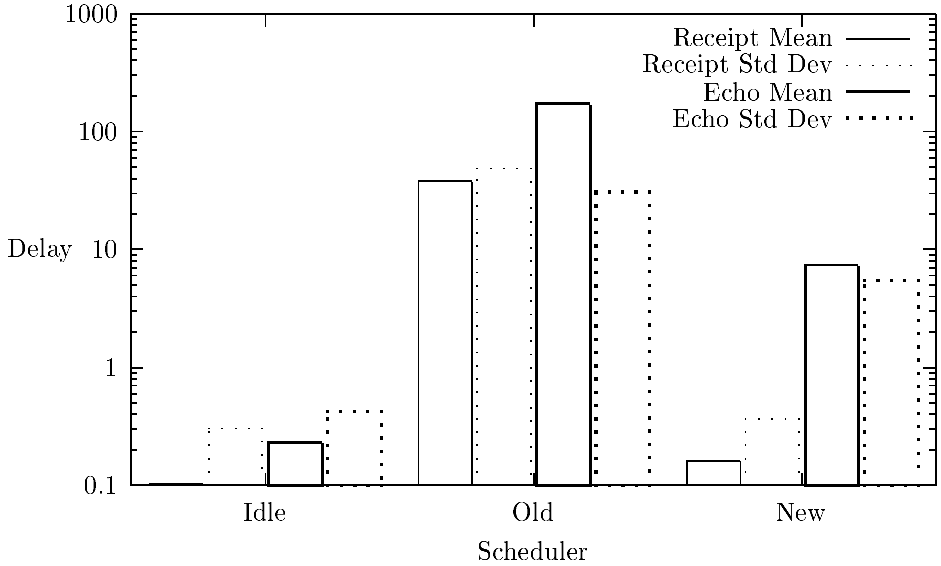

Figure 1: Scheduler Performance for Interactive Test

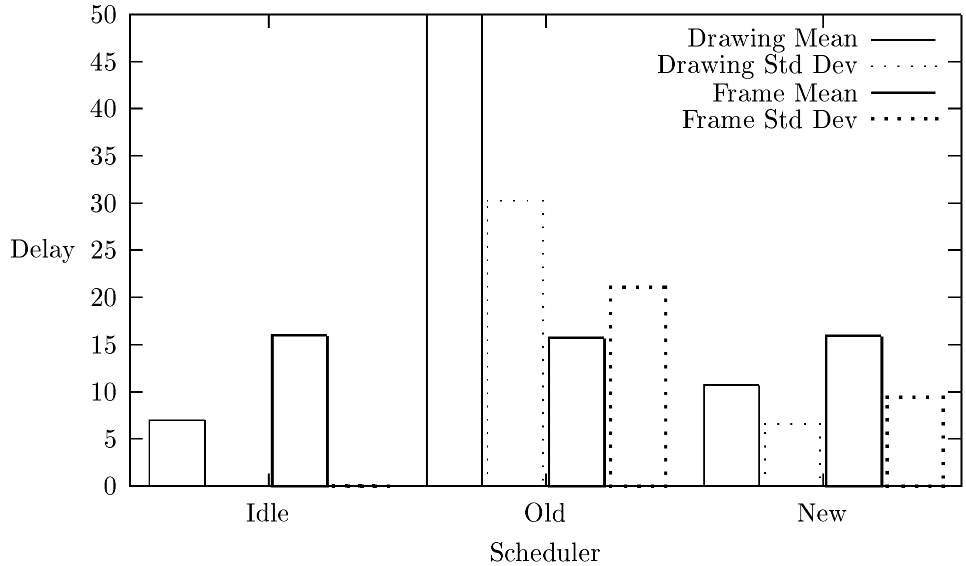

Figure 2: Scheduler Performance for Animation Test

7.3 Performance Measurement

X11perf is a tool used to measure X server performance. It is entirely synthetic, measuring repeated sequences of hundreds of primitive operations. The methods used to collect the data are relatively precise, and with some analysis and understanding, useful data can be extracted.

By inferring what effects can be expected from the scheduler changes, particular x11perf tests can be used to verify those effects. As x11perf is expected to run without competition from other applications, most of the dramatic changes seen above will be absent. One change which seemed promising was the effort to reduce calls to `select(2)`. The old scheduler would call `select(2)` whenever the client request buffer no longer contained a complete re-

Test	Old	New	New/Old
PolyPoint	4740000	4900000	1.03
NoOp	2130000	2680000	1.26

Table 2: X11perf Performance Comparison

	Old	New	New/Old
Xmark	24.9919	25.4732	1.02

Table 3: Xmark Comparison

quest. The new scheduler instead tries a `read(2)` call instead. This will return any pending data or `EWOULDBLOCK` if no data are available, if no data are ready, the server then waits in `select(2)`.

The effect of this change should be greatest for requests which execute quickly, and as seen in table 2 this is indeed the case. The units are operations per second.

Although the tests noted above did show measurable changes, for most of the test results, the two schedulers performed within 2% of each other. This shows that the scheduler changes have little effect on this performance measurement tool.

A standard synthetic benchmark, Xmark, has been created which performs a weighted geometric average of the x11perf test results. While it is more meaningless than most benchmarks, it has, nonetheless gained some popularity.

As the results in table 3 are generated from the x11perf numbers, the result is not surprising.

8 Kernel Timer Granularity

As measured above, the scheduling interval within the X server is not short enough to eliminate significant jitter in animation applications. The scheduling interval within the X server was set to 20ms with the knowledge that the usual Linux kernel timer interrupt was set to 10ms. Thus the operating system limits the ability of the X server to schedule clients precisely.

The 10ms interval has remained unchanged since it was chosen for the BSD Unix implementation for the Digital VAX machines [LMKQ89]. 100 interrupts per second was considered a negligible load on a machine of that era. With modern machines being somewhat faster, it seems reasonable to consider a higher resolution timer.

Fortunately, Linux provides a single constant which defines the frequency for timer interrupts. The kernel was rebuilt with a 1ms timer interval, and the X server was run with various scheduler intervals to measure the scheduler behavior as well as overall X server performance.

8.1 Interactive Test with Various Scheduler Intervals

For each scheduling interval, the performance of the interactive test was measured. The results are displayed in Figure 3. The mean delay between event generation and receipt is displayed (Receipt) with error bars indicating the standard deviation. The mean delay between event generation and the drawing of the line is also displayed (Echo). Again error bars indicate the standard deviation.

As the X server scheduling interval decreases, the latency between mouse motion and the echo on the screen decreases. Furthermore, the variation in echo latency values also decreases. There is no change in the event receipt latency because the X server checks for pending user input before processing each request.

8.2 Animation Test with Various Scheduler Intervals

For each scheduling interval, the performance of the animation test was measured. The results are displayed in Figure 4. The mean delay between issuing drawing requests and the display on the screen is displayed (Drawing latency) with error bars indicating the standard deviation. The mean time between the display of consecutive frames is displayed (Frame interval) with error bars indicating the standard deviation.

As the X server timer interval is made shorter, the deviation in the frame interval is reduced along with the drawing latency. Reducing the frame interval deviation will make animations appear smoother.

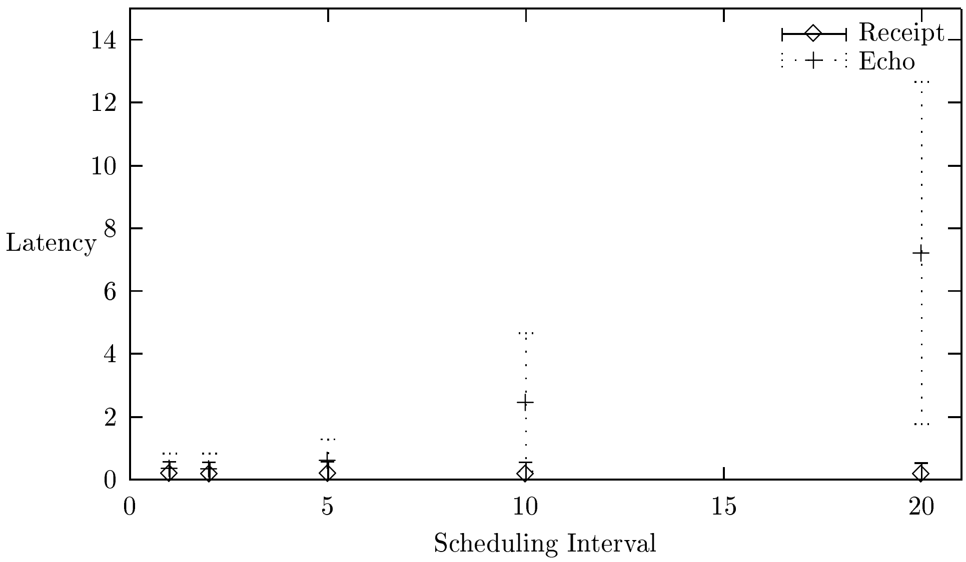

Figure 3: Effects of Changing Scheduler Interval on Interactive Test

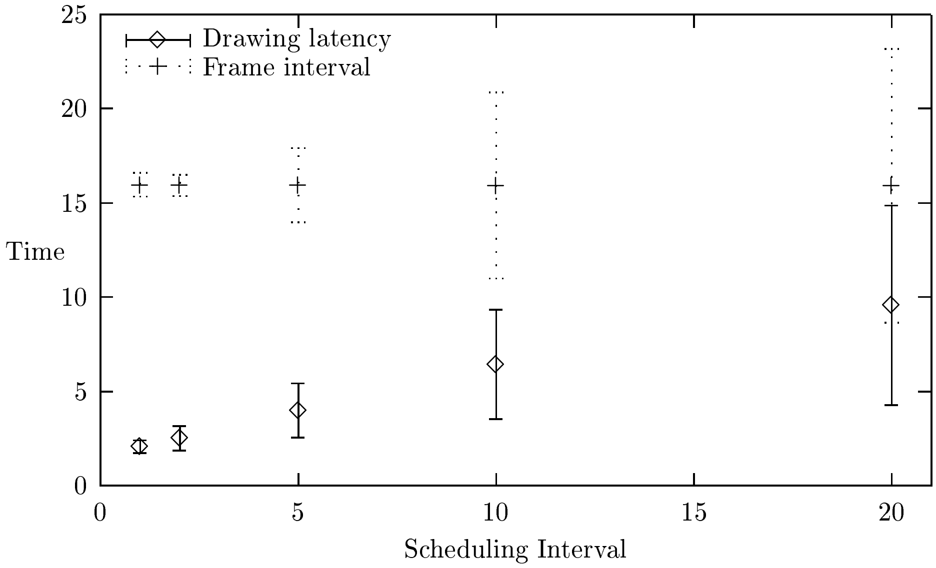

Figure 4: Effects of Changing Scheduler Interval on Animation Test

Reducing the drawing latency will improve synchronization with other media.

8.3 Performance Effects from Various Scheduler Intervals

Two effects should tend to decrease performance with shorter scheduling intervals. As the server uses a signal to duplicate the kernel clock inside the X server, increasing the frequency of signal delivery will increase the overhead of this process. Further, as the X server checks for other client activity by using `select(2)`, an increase in the scheduling frequency will increase the amount of time spent in `select(2)`.

The new scheduler includes two separate parameters

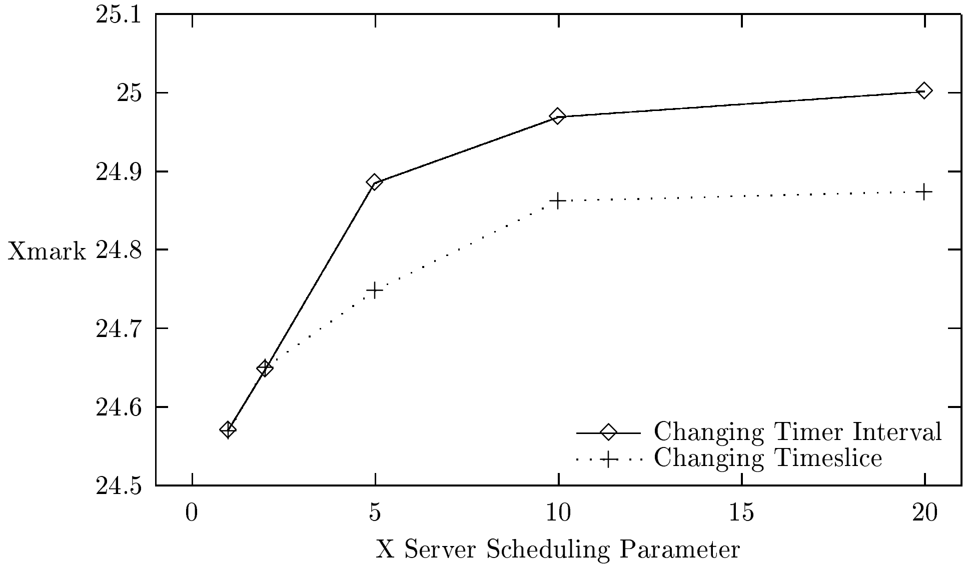

Figure 5: Effects of Changing Scheduler Parameters on Xmark Measurements

to control these two effects, the first is the timer interval and the second is the maximum timeslice granted to the client.

To gauge the overall impact of these two variables, x11perf was run with the X server set to use a a variety of timer intervals and maximum timeslice values. The results are displayed in Figure 5.

The effect of calling `select(2)` every millisecond is not very large in this X server. There is less than a 2% decrease in Xmarks when decreasing the time between `select(2)` calls from 20ms to 1ms.

While the overall effect is small, particular applications may see larger changes, depending on the balance between X server CPU and graphics accelerator usage. Operations which are limited by the graphics accelerator will show small performance impact, while operations limited by the CPU will show more.

Also evident is that most of the performance impact comes not from a high-resolution timer, but from the cost of `select(2)`. The new scheduling system dynamically increases the timeslice when running a single application, balancing the benefits of reducing system call overhead with providing accurate scheduling for multiple clients.

8.4 Performance Effects of a Kernel Timer Change

The kernel timer is set to a compromise between scheduling precision and the performance impact of additional timer interrupts. Increasing the timer frequency brings an associated increase in kernel processing as it reschedules processes more frequently, possibly context switching more often which affects TLB/cache performance.

X11perf was run with identical scheduling parameters with a kernel timer of 1ms and then 10ms, the resulting Xmark numbers appear in Figure 6. As the graph shows, the performance effect of an increase in kernel clock resolution is a decrease of between 1 and 1.5 percent.

8.5 The Kernel Timer Should Change

The measurements above demonstrate that the X server could provide significantly better scheduling if a higher resolution kernel clock were made available. The impact of raising the timer interrupt frequency, at least on X performance, is nominal given the performance of modern hardware.

Graphics hardware is in a unique situation, it provides output without generating interrupts. This

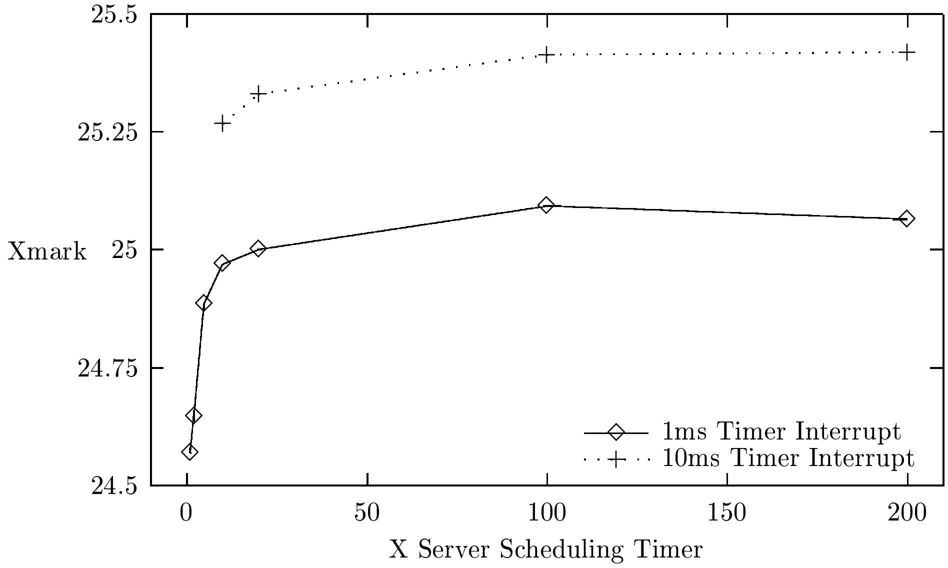

Figure 6: Effects of Changing Kernel Timer on Xmark Measurements

makes it difficult to schedule graphics activities precisely enough to avoid jitter even when the system is idle.

However, the performance impact of such a change needs to be addressed, even the minor 1 to 1.5 percent decrease in performance measured by this change. The measurements made here indicate that the kernel is spending additional cycles managing the increased timer interrupt rate. Perhaps the interaction between the timer interrupt and the upper level timer management code could be adjusted to avoid this overhead.

9 Remaining issues

There are still a few difficult problems which remain unsolved by this new scheduler.

9.1 Threading the X Server

Most X requests are short and execute rapidly, however there are a few core protocol requests and many extension requests which are not so well behaved. A straightforward solution to this problem is to create separate threads executing requests for each client, and then to build suitable locking mechanisms throughout the server to protect global data. MIT, Data General and Omron cooperated in 1991-1992 to build such a server [Smi92].

The result demonstrated that the rendering engine was a resource that every application needed to access for nearly every request, reducing the multi-threaded X server to a lock-step procession of threads waiting for the display hardware mutex. With multiple screens, a small amount of parallel execution would be possible.

9.2 Multiple Client Performance

The performance optimization of increasing the timeslice granted to a single application to reduce the system call overhead isn't currently done when multiple applications are running. This makes it likely that some requests will execute more slowly under the new scheduler than with the old, but only when more than one client is active.

9.3 Other Kernel Changes

To provide better support for user-mode scheduling, the kernel could make available an inexpensive copy of the system clock. One simple idea would be to create a shared segment containing a copy of the clock and make that mappable by user-mode programs. Additional information about system load might usefully be included in such a segment. Such a change would improve this scheduler by increasing the resolution of the clock, providing more accurate updates when the system is heavily loaded and eliminate the burden of frequent timer signal generation and reception.

A way of detecting when new data are available for the X server to read without the expense of `select(2)` would be useful. The X server constantly calls `select(2)` to see if any idle clients happen to have new requests pending. Perhaps the server could poll a local variable to determine whether select would return different information from the previous call. Such a variable could be set from a signal handler.

10 Conclusion

A simple scheduler, based on what information could be easily obtained by the user-mode X server, demonstrates some significant advantages over the original scheduler without negatively impacting performance. These changes are largely hidden from the user, who will only notice them by the absence of large delays when dealing with applications which flood the X server with requests.

11 Acknowledgments

I thank SuSE for encouraging me to work full time on X. This work was implemented during the X Hothouse event sponsored by SuSE at the Atlanta Linux Showcase in October 1999.

12 Availability

A previous version of this work has been incorporated into the 4.0 release of the X Window System from the XFree86 group. The current version will be available in the next public XFree86 release.

`http://www.xfree86.org`

References

[GCGW91] Tim Glauert, Dave Carver, Jim Gettys, and David P. Wiggins. X Synchronization Extension Protocol, Version 3.0. X consortium standard, X Version 11 Release 6, 1991.

[LMKQ89] Samual J. Leffler, Marshall Kirk McKusick, Machael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison Wesley, 1989.

[MKA+94] Joel McCormack, Phil Karlton, Susan Angebranndt, Chris Kent, Keith Packard, and Graeme Gill. X11perf - x11 server performance test program. Manual page, X11 Version 11 Release 6.4, 1994.

[MS96] Larry McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. In *Technical Conference Proceedings*, pages 279–284, San Diego, CA, January 1996. USENIX.

[SG92] Robert W. Scheifler and James Gettys. *X Window System.* Digital Press, third edition, 1992.

[Smi92] John Smith. The Multi-Threaded X Server. *The X Resource*, 1:73–89, Winter 1992.

[Tho78] K. Thompson. Unix implementation. *The Bell System Technical Journal*, 57(6):1931–1946, July-August 1978.