Implementing and Evaluating Jukebox Schedulers Using JukeTools

Maria Eva Lijding   Sape Mullender   Pierre Jansen

Fac. of Computer Science, University of Twente
P.O.Box 217, 7500AE Enschede, The Netherlands
E-mail: lijding@cs.utwente.nl

Abstract

Scheduling jukebox resources is important to build efficient and flexible hierarchical storage systems. JukeTools is a toolbox that helps in the complex tasks of implementing and evaluating jukebox schedulers. It allows the fast development of jukebox schedulers. The schedulers can be tested in numerous environments, real and simulated. JukeTools helps the developer to easily detect errors in the schedules. Analyzer tools create detailed reports on the behavior and performance of any of the scheduler, and provide comparisons between different schedulers.

This paper describes the functionality offered by JukeTools, with special emphasis on how the toolbox can be used to develop jukebox schedulers.

1.   Introduction

This paper presents *JukeTools*, a toolbox for designing, implementing, and testing jukebox storage systems. This toolbox resulted from research in scheduling jukebox resources and building a *hierarchical multimedia archive (HMA)*. However, the toolbox can also be used to evaluate different hardware architectures, caching policies and services offered by the storage system. In this paper we focus on using the toolbox to implement and evaluate schedulers.

A jukebox is a large tertiary storage device whose *removable storage media (RSM)*—e.g. CD, DVD, magneto-optical disk, tape—are loaded and unloaded from one or more drives by one or more robots. A jukebox can store large amounts of data in a cost-effective way, which makes it eminently suitable for applications that handle large amounts of continuous-media files, large databases and backups.

In order to use a jukebox effectively it is important to schedule the jukebox resources. On the one hand, a jukebox is not a random-access device: the RSM switching times are in the order of seconds or tens of seconds, which implies that multiplexing between two files stored in different RSM is many orders of magnitude slower than doing the same in secondary storage. On the other hand, the resources in the jukebox—robots, drives and RSM—are shared and require exclusive use, which creates the potential for resource-contention problems.

JukeTools allows the fast development of jukebox schedulers. Schedulers can be tested using different types of requests, caching policies and hardware. The verification functionality of the toolbox checks the validity of the schedules and helps the developer to detect errors. JukeTools is specially useful to detect resource contention problems.

A number of analyzer tools can produce detailed reports in the form of HTML pages, performance graphics and statistical data about major system components. Other analyzers combine the statistical data of multiple runs to compare the performance of schedulers under different load conditions. The log-files and intermediate reports are encoded in XML [13]. Therefore, new analyzers are easy to write based on XSLT [12] or XML-parsers. Also, additional details can be added to the log without affecting existing analyzers, because unknown entries are ignored.

We have implemented and compared six schedulers (in multiple variations) with JukeTools, both periodic and aperiodic. These schedulers cover all known real-time jukebox schedulers that deal correctly with the resource contention problem—Lau's aggressive and conservative strategies [7], first-come-first-serve (FCFS) [3]— and three new schedulers that we have developed—Promote-IT [9], the *jukebox early quantum scheduler (JEQS)* [8] and the *buffered jukebox early quantum scheduler (B-JEQS)*. The implementation of the old schedulers took only between one and three days, each.

2.   Toolbox Structure

Figure 1 gives an overview of the toolbox. The jukebox scheduler is at the core of the toolbox and also the primary component of the HMA. The HMA and the scheduler are presented in the next section.
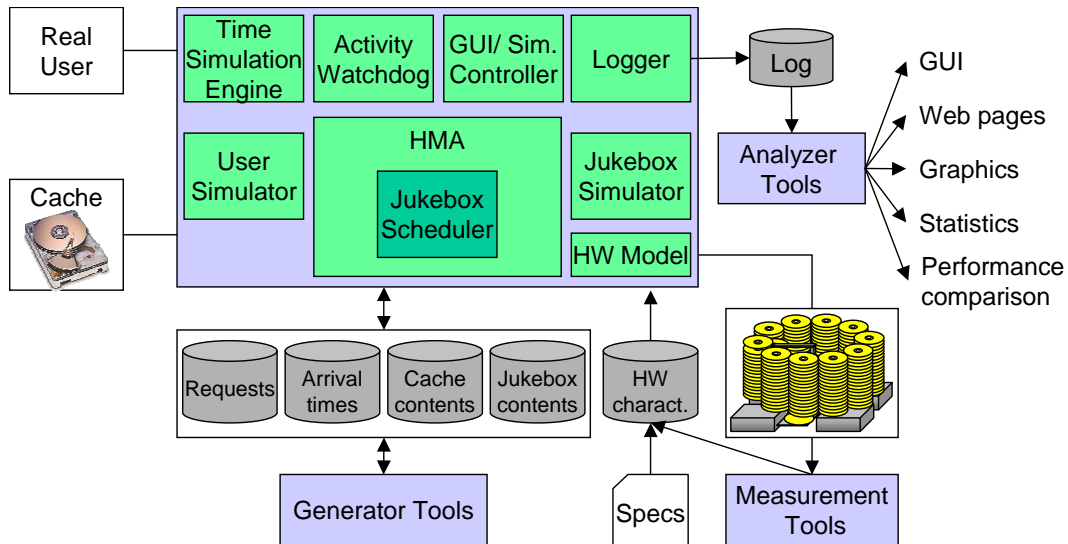
Figure 1. Architecture of JukeTools.

The toolbox supports simulated and real users to use simulated and real hardware. It hides the difference between 'simulated' and 'real' from the HMA components. Therefore, the developer can use the same code for the simulations and the operational system. This allows to thoroughly test the HMA components in a controlled environment before using them in operational systems (more benefits of this approach are presented in [2]). To narrow the gap between simulation and operational system even more, the toolbox uses a detailed analytical *hardware model* to simulate any type of jukebox hardware.

The *time simulation engine* speeds up the simulation by removing idle periods and maintains the virtual time of the system. The engine is presented in Section 5.

The requests for the HMA originate either from real users outside the toolbox or from the user simulator. The *user simulator* combines the data of a *request file* and an *arrival-times file* to generate the workload for the HMA. These files are created by *generator tools*. Generator tools also create synthetic *jukebox contents* and *cache contents*. The jukebox contents contain the location of each RSM in the jukebox and the directory of each RSM. The cache contents keep information about the data currently in the cache and map the data to file-system identifiers in secondary storage. For the HMA there is no difference between synthetic data and real data stored by itself during the operation of the system. This is another point that supports the transparent use of the HMA in a simulated or real environment.

The hardware model is strongly data-driven. It is built using the specifications provided in the *hardware-characteristics file*. The information of the file may corre-

spond to vendor specifications, the output of the *measurement tools* or the wishes of the toolbox user. The hardware model estimates the time needed to perform operations on the robots and drives. The estimates are used by the scheduler and the *jukebox simulator*. Section 4 presents more details.

JukeTools provides different ways to evaluate and monitor the operation of the system. The *logger* provides a log service to all the components in the toolbox. The log is later processed by *analyzer tools* to produce reports in the form of web pages, graphics, statistics and performance comparisons. More complex analyzers provide graphical representations of the schedules created during the execution of the system and details about the utilization of the jukebox resources.

Different components observe the execution of the HMA using the publisher-subscriber pattern [4]. The GUI offers different views of the running system and an attached simulation controller allows to pause, resume and stop the simulation. The *activity watchdog* uses the events generated by the HMA to detect possible deadlocks in the use of shared resources (more in Section 3).

Given that the bandwidth offered by the drives in a jukebox is generally much higher than the one required by the end users and that a jukebox does not provide random-access, the HMA stages data in a secondary storage cache from where it is delivered to the applications. The implementation of the secondary storage cache is outside the scope of the toolbox. At present we use a regular Linux file-system, but we are planning to use Clockwise [1] in the future to guarantee real-time access to the data in the cache.

The toolbox is implemented in Java, except for some functionality which is operating system dependent. The drive controllers use the Java Native Interface (JNI) to call C functions on Linux in order to open and close the drives and get drive specific information. Additionally, the analyzer tools use gnuplot to generate the graphics. The toolbox and the implemented schedulers are available through the authors.

3. Hierarchical Multimedia Archive

The HMA can serve complex requests for the real-time delivery of any combination of media files it stores. A request can consist of multiple streams and non-streamed data that are synchronized sequentially or concurrently in arbitrary patterns. Such requests can originate from any system that needs to combine multiple, separately stored media files into a continuous presentation. Examples are queries to a multimedia database to assemble a TV documentary, or a computer generated play list for a huge library of music videos and advertisement that produces an MTV-like program.

The HMA can also be used for the more simple case of a Video-on-Demand (VoD) application where the requests are generally for only a single media file—a movie—to be played from beginning to end. The capacity of the HMA to serve complex requests is restricted by the capacity of its current scheduler. The HMA is for example restricted to VoD-like requests when the jukebox early quantum scheduler is used.

The HMA was designed to guarantee high quality of service to the users, but it can also be used to build applications offering more relaxed or no quality of service. In the original usage scenario, once the system accepts and confirms a request from a user, it is committed to provide the service requested by the user. The confirmation includes the starting time assigned to the request. The user can start consuming the data at the starting time, with the system's guarantee that the flow of data will not be interrupted.

Figure 2 shows the architecture of the jukebox scheduler. The cache manager filters out the parts of incoming requests that refer to data that is already in the cache or scheduled for staging. The schedule builder schedules the filtered requests on-line, re-computing the schedule whenever a request arrives. It generates a new schedule to replace the currently *active schedule*. The dispatcher uses the active schedule to send commands to the jukebox controller, which move RSM and stage data into secondary storage.

In the case of real-time schedulers the schedule builder guarantees that including the new request does not lead to missed deadlines and the dispatcher guarantees that the commands are sent to the controller in time. The dispatcher
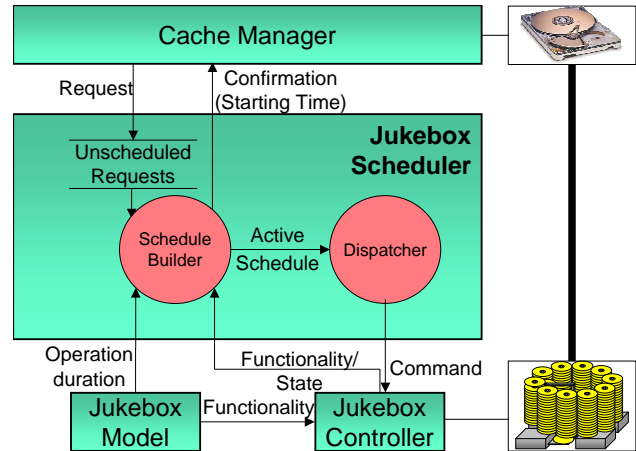

Figure 2. Jukebox-scheduler architecture.

may modify the schedules as long as no task in the schedule is delayed and the sequence and resource constraints are respected (see [9] for a discussion on the benefits of early dispatching).

The HMA is highly modular and the interfaces between the components are clearly defined. The schedule builder, the dispatcher and the hardware controllers run as independent threads. The jukebox controller operates as a schedule verifier, because it only performs valid commands. It determines if a command is valid using information about the location of each RSM in the jukebox and the state of each device. This information cannot be modified by the scheduler. The jukebox controller can detect illegal commands that request to: unload an empty drive, load a loaded drive, read data from an empty drive, unload an RSM different from the one loaded or load the same RSM in two different drives. The controller also reports if the commands are executed late or if the execution time is different from the time that was estimated by the scheduler. The latter is specially important when using real hardware.

The activity watchdog detects more complex conflicts in the use of the resources as deadlocks in the controller queues. The watchdog runs as an independent thread and listens to the events generated by the HMA components. The watchdog can detect deadlocks in the HMA components, if the components publish the beginning and end of an operation that may lead to a deadlock, and indicate the maximum time to perform the operation.

3.1. Requests

The requests consist of a deadline and a set of request units u_{ij} for individual files, or part of files. The requests can represent any kind of static temporal relation between

the request units. Formally we represent a request r_i with l_i request units as:

$$r_i = (\tilde{d}_i, asap_i, maxConf_i, \{u_{i1}, u_{i2}, \ldots, u_{il_i}\})$$
$$u_{ij} = (\Delta\tilde{d}_{ij}, m_{ij}, o_{ij}, s_{ij}, b_{ij})$$

The deadline $\tilde{d}_i$ of the request is the time by which the user must have guaranteed access to the data. The flag $asap_i$ indicates if the request should be scheduled as soon as possible. The maximum confirmation time $maxConf_i$ is the time the user is willing to wait in order to get a *confirmation* from the system, which indicates if the request was accepted or rejected. The relative deadline of the request unit $\Delta\tilde{d}_{ij}$ is the time at which the data of the request unit should be available, relative to the starting time of the request. The other parameters of the request unit m_{ij}, o_{ij}, s_{ij} and b_{ij} represent the RSM where the data is stored, the offset in the RSM, the size of the data, and the bandwidth with which the user wants to access the data, respectively.

The confirmation to the user indicates if the request is accepted or rejected. If the request is accepted, the confirmation contains the starting time st_i assigned to the request. The starting time must be less or equal to the deadline of the request ($st_i \leq \tilde{d}_i$). If the request is ASAP the scheduler tries to find the earliest value of st_i that will allow it to accept the request. The system must provide a confirmation before $maxConf_i$.

4. Jukebox Model and Simulation

We use a model of the hardware to predict the time that the system will need for operations on robots and drives. We use this model to build the schedules and time the jukebox simulator. We have validated our jukebox model against our actual hardware.

The scheduler basically needs to know the time required to load an RSM in a drive, read data from the RSM and unload it. Computing these times is not straightforward. Many factors are important as the type of RSM, the drive, the jukebox robotics, the number of robots in the jukebox and the location of the shelves and drives in the jukebox. Our hardware model has separate sub-models for the RSM, the drives, the robots and the jukebox. Together, the sub-models can describe any type of jukebox architecture.

The hardware model can handle jukeboxes where the drives are not identical. This allows the toolbox to deal with situations in which the hardware of the jukebox goes through gradual upgrades. The model can also handle jukeboxes with multiple robots. Each robot has an associated functionality and scope. Its function can be loading, unloading or both. The scope of a robot is given by the set of drives and shelves it can serve (see [10] for more details about the model).

A goal of the jukebox simulator is to provide the same execution pattern during each execution of a simulation, so that the results are reproducible. If we run a simulation with the same input we want to obtain the same performance results. Therefore, the jukebox simulator assumes that the execution of the operations take exactly the time indicated by the model.

The simulator uses the 'simulation model' proposed by Ruemmler et al. [11] when relevant. The drive simulator, for example, keeps track of the last time that the drive performed a read to decide if a spin-up is needed. However, we do not consider it relevant to know the exact rotation time in an access to data on a disc, because it is very small compared to the other components of the access. Additionally, the exact rotation time varies if a task is dispatched with a small time difference (e.g. one millisecond), therefore, the resulting execution will not be the same.

At present the implementation of the hardware model can handle any type of optical and magneto-optical jukebox. We are mainly concerned with these type of jukebox technology, because discs are better suited for random access than tapes, and can be loaded and unloaded faster. The implementation can easily be extended to include other type of storage media, drives and jukebox hardware. A good starting point to include magnetic tapes is to implement the model to estimate the locate-time on serpentine tapes provided by Hillyer et al. [5] and the benchmark methodology presented by Johnson et al. [6].

5. Time Simulation Engine

The *time simulation engine* is an event simulator with the capacity to remove waiting time from the simulation. Waiting time in a simulation are the periods where a real system would wait for hardware operations or new user requests, and the system is not busy performing computations.

The engine manages the virtual time of the system, delivers timed wake-up events and provides thread synchronization through semaphores. Threads register themselves with the engine at creation time. By tracking the state of the threads through the use of the semaphores the engine can detect and eliminate waiting times.

A thread can be in one of three states: *active*, *blocked* or *sleeping*. An active thread performs computations and needs to run in real time. A blocked thread is waiting for an event from an active thread, e.g. waiting on a semaphore. A sleeping thread is waiting for its wake-up event, e.g. to simulate waiting for the robot to finish moving.

As long as at least one thread is active, the virtual time advances in real time. When all threads are blocked or sleeping, the time simulation engine advances the virtual time to the time of the next event.

The time simulation engine is no thread scheduler—thread scheduling is performed by the Java Virtual Machine (JVM). The engine runs in a thread with the highest priority so that it always gets the right to execute when ready. Bosch et al. [2] use the concept of real and virtual time in a thread scheduler.

6. Conclusions

We have presented a toolbox to develop, evaluate and compare jukebox schedulers. The toolbox allows the developer to concentrate on the topics relevant for scheduling and abstract from secondary issues. JukeTools helps to detect inconsistencies in the use the jukebox resources and missed deadlines. It also provides detailed reports on the performance of the scheduler, which can be used to detect bottlenecks and inefficiencies. The toolbox is flexible and can be configured easily to simulate numerous hardware architectures, scheduling policies and user behavior. Therefore, it provides an ideal framework to evaluate and compare different schedulers. JukeTools provides an environment that makes simulation transparent to the developer to the extent that the same code can be used for simulations and operational systems. So far, we have used the toolbox to implement and compare six different schedulers, plus multiple variations.

The toolbox can also be used to evaluate different hardware architectures, caching policies and services offered by the storage system. Furthermore, making some modifications to the semantics of the components and the interfaces, the toolbox can also be used to implement real-time schedulers for other environments as secondary storage, and manufacturing systems.

Acknowledgments

References

[1] P. Bosch. *Mixed-media file systems*. PhD thesis, University of Twente, June 1999.

[2] P. Bosch and S. J. Mullender. Cut-and-paste file-systems: Integrating simulators and file-systems. In *USENIX Annual Technical Conference*, pages 307–318, 1996.

[3] S.-H. G. Chan and F. A. Tobagi. Designing hierarchical storage systems for interactive on-demand video services. In *Proc. of IEEE Multimedia Applications, Services and Technologies*, June 1999.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass., 1995.

[5] B. K. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proc. of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 170–179, May 1996.

[6] T. Johnson and E. L. Miller. Benchmarking tape system performance. In *Proc. of Joint NASA/IEEE Mass Storage Systems Symposium*, March 1998.

[7] S.-W. Lau and J. C. S. Lui. Scheduling and replacement policies for a hierarchical multimedia storage server. In *Proc. of Multimedia Japan 96, International Symposium on Multimedia Systems*, March 1996.

[8] M. E. Lijding, F. Hanssen, and P. G. Jansen. A case against periodic jukebox scheduling. Technical Report TR-CTIT-02-47, Centre for Telematics and Information Technology, University of Twente, November 2002.

[9] M. E. Lijding, P. G. Jansen, and S. J. Mullender. A flexible real-time hierarchical multimedia archive. In *Proceedings of the Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems, (IDMS/PROMS)*, volume 2515 of *Lecture Notes in Computer Science*, pages 229–240, Coimbra, Portugal, November 2002. Springer-Verlag, Berlin.

[10] M. E. Lijding, S. J. Mullender, and P. G. Jansen. A comprehensive model of tertiary-storage jukeboxes. Technical Report TR-CTIT-02-41, Centre for Telematics and Information Technology, University of Twente, October 2002.

[11] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[12] W3C. *XSL Transformations (XSLT) 1.0*, November 1999.

[13] W3C. *Extensible Markup Language (XML) 1.0*, second edition, October 2000.