Efficient Guaranteed Disk Request Scheduling with Fahrrad

Anna Povzner, Tim Kaldewey, Scott Brandt, Richard Golding†, Theodore M. Wong†, Carlos Maltzahn

Computer Science Department, University of California, Santa Cruz
†IBM Almaden Research Center, San Jose, CA

European Conference on Computer Systems 2008
Outline

- Introduction
- The Fahrrad Scheduling
- Dealing With Unqueued Requests
- Evaluation
- Conclusion
Introduction

- Like a CPU scheduler, the basic goal of a real-time disk
  - I/O scheduler is to provide timeliness guarantees
  - hard real-time, soft real-time, and best-effort

- good request scheduling
  - Sequential I/O accesses v.s. random accesses

- real-time I/O scheduler
  - not just guaranteed but also good performance
  - isolate request streams
Introduction

- **General-purpose applications**
  - I/O performance: throughput
- **Real-time application**: latency and throughput
  - latency is bounded by a reservation granularity
- **throughput is challenging for 4 reason**:
  - individual disk requests are non-preemptible
  - I/O request times are stateful (previous request, I/O streams)
  - I/O times are partially non-deterministic
  - best-, average-, and worst-case I/O times vary much
Introduction

- Hard throughput guarantees
  - worst-case assumptions about request times
  - the ability to reserve ~0.01% of the maximum achievable disk throughput

- The Fahrrad real-time disk I/O scheduler
  - based on disk time utilization reservations
    - easily reservable
    - easily manageable
    - encapsulate knowledge about application I/O behavior
The Fahrrad Scheduling

- I/O reservations are made via a broker
  - The broker **decides if a reservation is feasible** (and **allowed**) and informs both the requester and the I/O system of successful reservations

- a requester specifies its **desired throughput and/or latency** and its **expected I/O behavior** to the broker

- The broker **translates these into the utilization and granularity** required to support the desired throughput and latency given the application I/O behavior

- Where nothing is known about the I/O behavior, we can assume the worst-case
The Fahrrad Scheduling

- Latency is a combination of
  - the delay imposed in the scheduler
  - the delay caused by applications queueing up requests

- an application I/O requests will be queued no longer than the application-specified reservation granularity (period), bounding the latency imposed by the scheduler

- In Fahrrad, reservations are associated with I/O streams
The Fahrrad Scheduling

- Fahrrad extends RBED to disk I/O scheduling
  - allows applications to reserve utilization
  - specify deadlines at which the reservation must be met
- Fahrrad
  - I/O request dispatching is based loosely on EDF
  - similarly implements the Resource Allocation/Dispatching (RAD) scheduling model
    - *How much* resource to allocate to a process?
    - *When* to provide the process those allocated resources?
- 2+1 layer:
  - resource allocation (*rate* and *deadline*)
  - dispatching (chooses which process to execute)
  - Reordering (I/O request *ordering*)
The Fahrrad Scheduling

- **Fahrrad Scheduler Theory**
  - the worst-case request time (WCRT) bounded by the worst-case seek time of the device + the maximum rotational delay + plus the time required to transfer the data (use constant WCRT, 4K)

**Theorem 1.** Given a set of periodic tasks $T_i$ with period $p_i$ consisting of jobs $J_{i,j}$, each consisting of a stream of $m_{i,j}$ non-preemptible I/O requests $R_{i,j,k}$, each of which takes $\alpha_{i,j,k} \leq WCRT$ such that $\forall i, j \left( \sum_{k=1}^{m_{i,j}} \alpha_{i,j,k} \leq e_i \right)$, Earliest Deadline First (EDF) will determine a feasible schedule of I/O requests, as long as

$$U = \sum_{i=1}^{n} u_i + \frac{WCRT}{\min_{1 \leq l \leq n}(p_l)} \leq 1$$

- $n$: number of I/O streams in the system
- $T_i$: task which corresponds to an I/O stream
- $u_i$: disk time utilization of task $T_i$
- $p_i$: period of task $T_i$
- $e_i$: budget of each job of $T_i$, $e_i = u_i \cdot p_i$
- $J_{i,j}$: job of task $T_i$
- $r_{i,j}$: release time of job $J_{i,j}$
- $d_{i,j}$: deadline of job $J_{i,j}$
- $m_{i,j}$: number of I/O requests in job $J_{i,j}$
- $R_{i,j,k}$: I/O request of job $J_{i,j}$
- $\alpha_{i,j,k}$: actual execution time of request $R_{i,j,k}$
- $\rho_{i,j,k}$: (micro-)release time of request $R_{i,j,k}$
- $\delta_{i,j,k}$: (micro-)deadline of request $R_{i,j,k}$
The Fahrrad Scheduling

- Guaranteeing utilization
  - If \( m_{i,j} \) requests per period
  - Request service times are not known
  - Only be known after the request has completed
  \[ \sum_{k=1}^{m_{i,j}} \alpha_{i,j,k} = e_i \]

**Theorem 2.** Given a set of tasks \( T_i \) consisting of jobs \( J_{i,j} \) with budget \( e_i \), each job consisting of a series of requests \( R_{i,j,k} \) with actual execution time \( \alpha_{i,j,k} \leq \text{WCRT} \) known immediately after completion of the request, in order to guarantee the budget \( e_i \) in each period the scheduler must reserve \( u_i = u_i + \text{WCRT} / p_i \).
The Fahrrad Scheduling

- **Basic Fahrrad Scheduler**

![Diagram of Fahrrad architecture]

**THEOREM 3.** The number of seeks $S$ required to process $n$ streams of requests is $S \leq \sum_{i=1}^{n} (s_i + 2)$, where $s_i$ is the number of seeks required to process the requests of stream $i$ when handled in isolation, i.e. without interference from requests belonging to any other stream.
Dealing With Unqueued Requests

- To provide good guarantees
  - hold onto reservations as long as possible
    - a stream has the greatest possible chance of using its reservation regardless of when its requests arrive
    - by holding empty slots in the DSS for tasks that do not have enough requests queued up

- Empty slots have the potential to negatively impact
  - unqueued requests may cause extra seeks as the head moves
  - an empty slot due for execution is the disk equivalent of a task blocking itself during its period
Dealing With Unqueued Requests

- make good guarantees
  - hold empty slots as long as possible
  - avoid the overhead from the extra seeks needed to seek between
    - the requests of the stream which now has an empty slot
    - and those of some stream with a filled slot we would prefer to immediately expire the slot of the offending task

- While extra seeks are inevitable
  - when the bottleneck stream is using less disk time than it reserved, it will force the scheduler to hold empty slots until they expire
  - prevents the scheduler from extending the deadline
Evaluation

- Fahrrad prototype is implemented as a loadable block-device driver for the Linux 2.6.17 kernel
  - The driver sits on top of an underlying disk device and exports a block device named /dev/fahrrad
  - All streams share the same underlying device
    - A user-level program makes reservations via an ioctl() call
    - An fcntl() call is used to associate an I/O stream with a reservation

- Hitachi Deskstar DJNA-371350
  - 13.5 GB 7200 RPM IDE drive
  - average seek time : 8.5 ms
Evaluation

Sequential workloads

Figure 3: Performance of four sequential request streams as the period of stream 4 changes, while periods of other streams remain constant. Each stream reserves 20% of disk time. Results are the average of 10 runs, including error bars (too small to be visible).
Evaluation

- Sequential workloads

Figure 4: Performance of four sequential streams with fixed periods as the period of the HRT stream changes. Each stream reserves 20% disk utilization. AFAP refers to sequential streams that queue requests as fast as possible. Results are the average of 10 runs.
Evaluation

Non-sequential workloads

Figure 5: Performance of 4 sequential real-time streams and one best-effort stream as the period of the HRT stream changes. Each stream reserves 20% of disk time. AFAP refers to sequential streams that queue requests as fast as possible. Results are the average of 10 runs.
Evaluation

Workloads with latency requirements

**Figure 7**: Cumulative distribution of request response times of the random HRT stream. The HRT stream reserves 50% of disk time, and it runs with one random bursty stream on background. Each line represents different period reservation for the HRT stream.
Evaluation

- Comparison with a best-effort I/O scheduler

**Figure 6:** Performance of 4 semi-sequential real-time streams and one best-effort stream as the period of the HRT stream changes. Each stream reserves 20% of disk time. AFAP refers to sequential streams that queue requests as fast as possible. Results are the average of 10 runs.
Evaluation

Figure 8: Behavior of mixed workload during 500 seconds, with and without Fahrrad. Points are the average for 5-second intervals.
Conclusion

- The Fahrrad disk I/O scheduler
  - provides correct real-time scheduling within a single scheduler
  - combination of hard and soft real-time I/O streams

- basic scheduling algorithm
  - uses EDF internally
  - adding an ordering mechanism: DSS

- The implementation includes most of the features presented above including
  - request queues, the DSS, micro-deadline adjustment, early deadline extension, and slot swapping
  - The experiments we have run using this implementation show that the driver delivers excellent performance