Single Machine Scheduling

and Its Applications to Web Transmission

Ye Xia

Department of EECS

University of California at Berkeley

Model

- **Jobs (objects)** $1, 2, ..., n$
- **To be transmitted by a single server with capacity** μ
- **Object parameters (e.g. object** i**)**
  - r_i: **release time. We assume** $r_i = 0$ **for all** i**. Otherwise, the problem becomes much more difficult in most cases.**
  - s_i: **size**
  - p_i: **processing (transmission) time;** $p_i = s_i/\mu$
  - C_i: **completion time**
  - d_i: **due date (deadline)**
  - T_i: **tardiness;** $T_i = \max\{C_i - d_i, 0\}$

Problem Definition

- **Find an optimal processing schedule**
- **Examples of traditional optimality measures (minimization problems)**
  - **mean completion time** $\bar{C}$
  - **weighted completion time** $\sum_{i=1}^{n} \alpha_i C_i$, **where** $\alpha_i > 0$
  - **mean tardiness** $\bar{T}$
  - **maximum tardiness** T_{max}
  - **number of tardy jobs** $n_T = \sum_{i=1}^{n} \mathbf{1}\{C_i > d_i\}$

Regular Measures

Definition 1 ***A*** *regular measure* R ***is one that is a monotone function of completion times. That is,*** R ***is a function of*** $C_1, C_2, ..., C_n$ ***such that*** $C_1 \leq C_1', C_2 \leq C_2', ..., C_n \leq C_n'$ ***together implies*** $R(C_1, C_2, ..., C_n) \leq$ ***(or*** $\geq$***)*** $R(C_1', C_2', ..., C_n')$***.***

Examples of Regular Measures: $\bar{C}, \bar{L}, L_{max}, \bar{T}$, **and** T_{max}

Fact 1 ***For a*** *regular measure* R***, there exists an optimal work conserving schedule.***

Fact 2 ***For a*** *regular measure* R***, no improvement may be gained in the optimal schedule by allowing preemption.***

For regular measures, optimal schedule $\Longleftrightarrow$ *optimal sequencing*

Some Simple Schedules - Polynomial Problems

1. SPT (Shortest Processing Time) schedule

   Fact 3 SPT *(Shortest Processing Time) schedule minimizes* $\bar{C}$, $\bar{W}$, $\bar{L}$, *and* $\bar{N}_u$.

   Fact 4 *If all jobs have the same due dates,* SPT *schedule minimizes* $\bar{T}$.

   Fact 5 *If it is impossible for any job to be on time in any sequence, then* $\bar{T}$ *is minimized by* SPT *sequencing; if* SPT *yields no jobs on time, then it minimizes* $\bar{T}$.

Some Simple Schedules - Cont

2. WSPT (Weighted Shortest Processing Time) schedule

   Fact 6 *The* weighted completion time *measure* $\sum_{i=1}^{n} \alpha_i C_i$, *where* $\alpha_i > 0$ *for all* i, *is minimized by the sequence*

$$\frac{p_{i(1)}}{\alpha_{i(1)}} \leq \frac{p_{i(2)}}{\alpha_{i(2)}} \leq \ldots \leq \frac{p_{i(n)}}{\alpha_{i(n)}}$$

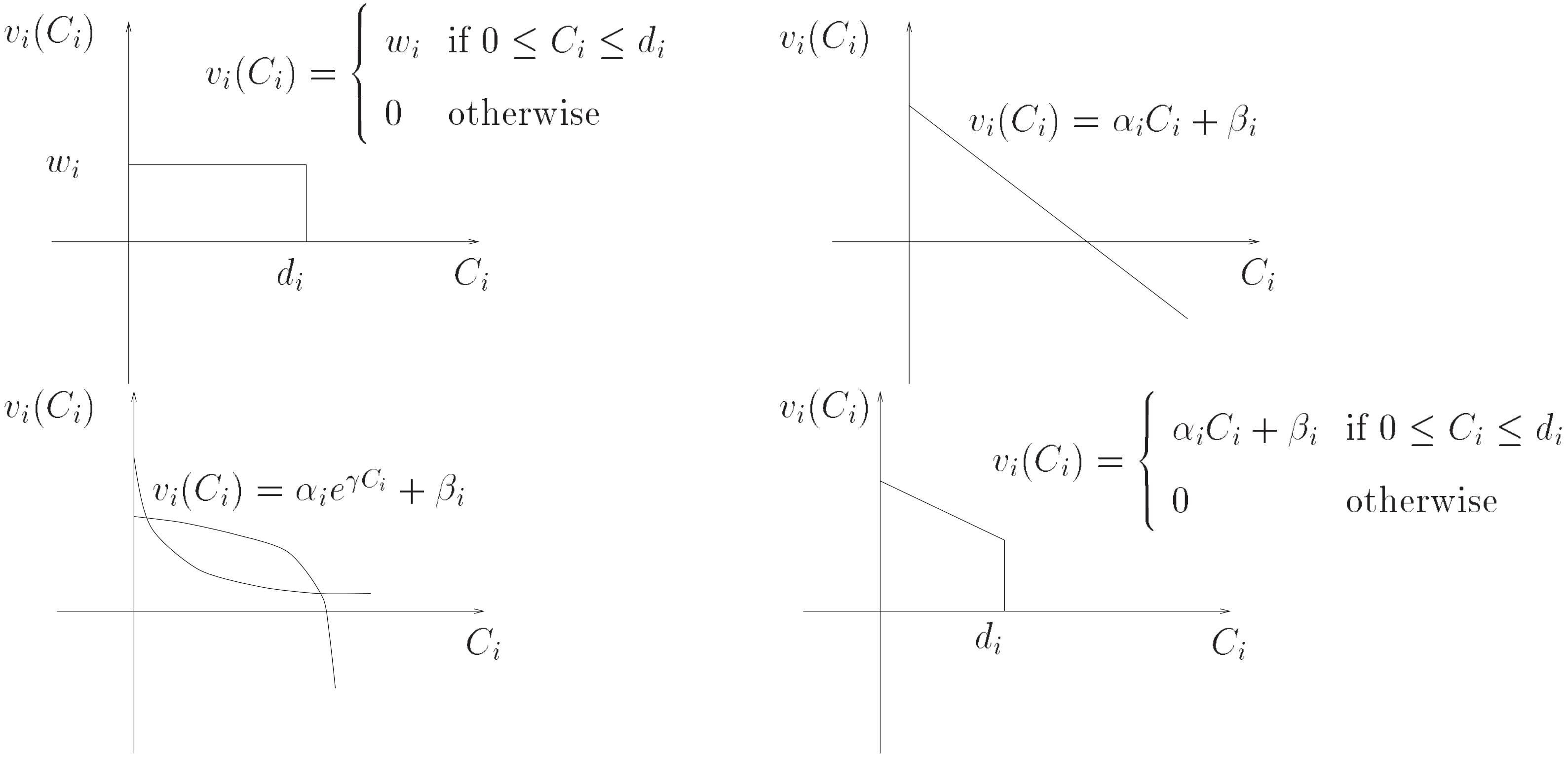
3. EDD (Earliest Due Date) schedule

   Fact 7 *There exists a schedule without tardy jobs iff the* EDD *schedule yields no tardy jobs.*

   Fact 8 EDD *schedule minimizes* L_{max} *and* T_{max}.

Utility Maximization

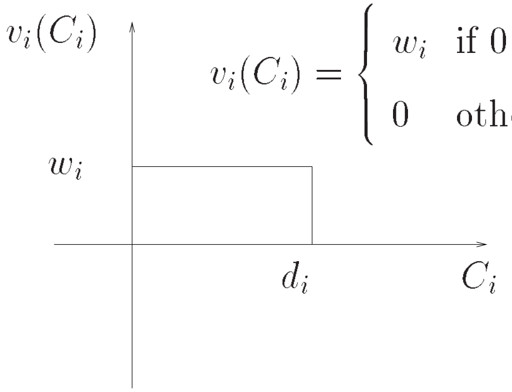
- **Define utility functions, $v_i(C_i)$, where C_i is the completion time.**
- **Find a schedule to maximize $\sum_{i=1}^{n} v_i(C_i)$.**



Some Results from Complexity Theory

- Due to "on-line" nature, we need polynomial algorithms
- For NP-hard problems, we have the following choices
  - "smart" enumeration techniques: dynamic programming, branch and bound, integer programming; with the hope that most cases can be solved in polynomial time.
  - pseudo-polynomial algorithms (ordinary NP-hard problems)
  - fully polynomial approximation schemes that guarantee worst case relative performance (ordinary NP-hard problems)
  - polynomial time heuristic algorithms with no performance bound

Utility Function 1 - Step Function



$$v_i(C_i) = \begin{cases} w_i & \text{if } 0 \leq C_i \leq d_i \\ 0 & \text{otherwise} \end{cases}$$

- **NP-hard in the ordinary sense**
- **Pseudo-polynomial algorithm based on dynamic programming exists; complexity $O(nP)$, where $P = \Sigma_{i=1}^{n} p_i$.**
- **Fully polynomial approximation algorithm exits with complexity $O(n^2/\epsilon)$.**

Step Function - Dynamic Programming Solution

- **Due to Lawler and Moore, 1969**
- **Assume jobs** $1, 2, ..., n$ **are in EDD order. For each job** i**, we need to decide whether to process it.**
- **Let** $P = \sum_{i=1}^{n} p_i$**. Assume** p_i**'s are integer-valued.**
- **Let** $f(j,t) =$ **the maximum gain for the first** j **jobs, subject to the constraint that job** j **is completed (including determining not to process** j**) no later than time** t**.**
- **Let** $c(j,t) =$ **the completion time of** j **for the schedule that achieves** $f(j,t)$**. It should be updated in conjunction with** $f(j,t)$**.**

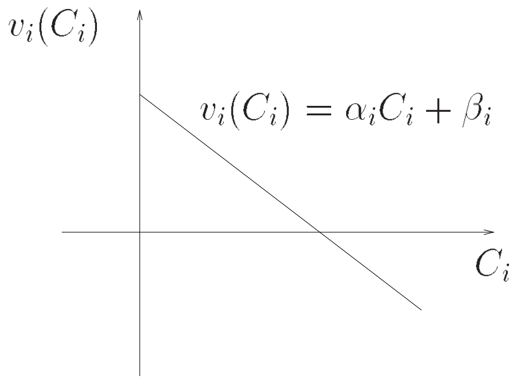
$$f(0,t) = 0, \qquad (t \geq 0)$$

$$f(j,t) = -\infty, \qquad (j = 0, 1, ..., n; t < 0)$$

$$f(j,t) = \max \left\{ \begin{array}{l} f(j, t-1) \\ w_j + f(j-1, t-p_j) \quad \textbf{if } p_j + c(j-1, t-p_j) \leq d_j \\ f(j-1, t) \end{array} \right\} \qquad (j = 1, 2, ..., n; t \geq 0)$$

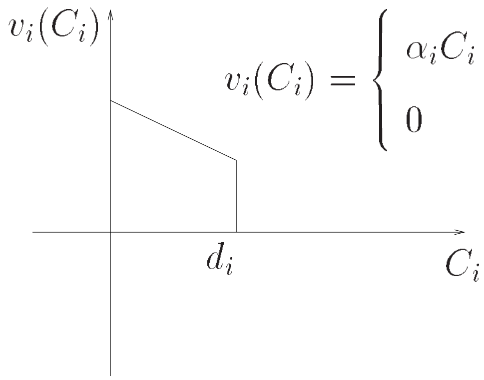
- $f(n, P)$ **is the optimal value for the problem.**
- **The time complexity is** $O(nP)$**. This is a pseudo-polynomial solution.**

Utility Function 2 - Linear Function



- **Optimal schedule is WSPT: order objects in increasing order of α_i/p_i.**
- **Complexity is $O(n \log n)$.**
- **Optimal schedule does not depend on β_i's.**
- **Drawback: tardy objects are penalized.**
- **This utility function has many desirable features.**

Utility Function 3 - Truncated Linear Function



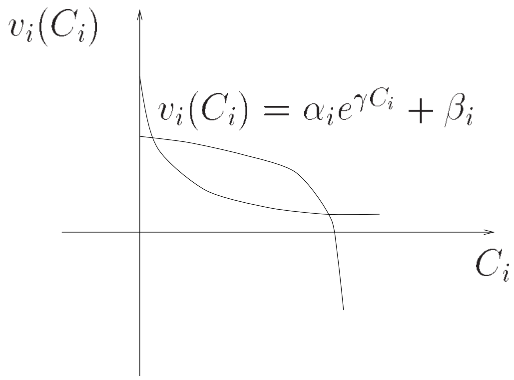
$$v_i(C_i) = \begin{cases} \alpha_i C_i + \beta_i & \text{if } 0 \leq C_i \leq d_i \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha_i \leq 0$$

Conjecture 1 ***The utility maximization problem is NP-hard in the strong sense.***

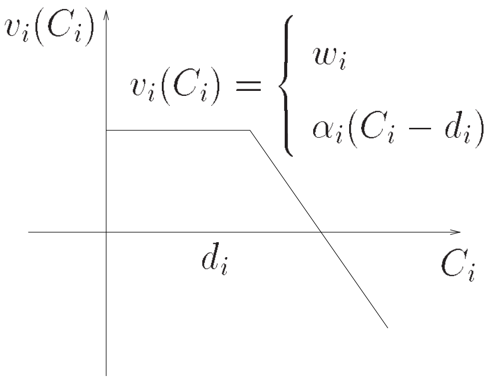
- **At least NP-hard in the ordinary sense. (Try $\alpha_i = 0$)**
- **When $\alpha_i < 0, d_i = -\beta_i/\alpha_i$ for all i, and all jobs are required to be completed before their due dates, the problem is strongly NP-hard.**

Utility Function 4 - Exponential Function



- **Depending on sign of γ, there are two family of functions.**
- **The optimal schedule is to order** $\frac{\alpha_i e^{\gamma p_i}}{1-e^{\gamma p_i}}$
  - **in increasing order when** $\gamma < 0$
  - **in decreasing order when** $\gamma > 0$
- **Has many similar features as linear utility function.**

Utility Function 5



$$v_i(C_i) = \begin{cases} w_i & \text{if } 0 \le C_i \le d_i \\ \alpha_i(C_i - d_i) + w_i & \text{otherwise} \end{cases}$$

- The maximization version of the weighted tardiness problem.
- NP-hard in the strong sense.
- If all α_i's are equal, it is NP-hard in the ordinary sense and can be solved in pseudo-polynomial time.
- Many algorithms are available for this problem.

Summary of Utility Functions and Look Ahead

- No work has been done on mixed utility functions: linear and step functions. That problem is at least NP-hard in the ordinary sense.
- Linear utility function is very nice.
  - Generalized SPT schedule
  - Single control variable
  - Even with a family of fairly complex precedence constraints, the time complexity can still be $O(n \log n)$.
  - Good performance when processing times are random

Precedence Constraints

- From now on, we will focus on linear utility function.
- Precedence constraints are represented by directed acyclical graph (DAG).
- For the case of general precedence constraints, the problem is strongly NP-hard.

Fact 9 ***Algorithms with complexity*** $O(n \log n)$ ***exist, for the class of constraints represented by series-parallel DAG.***

Examples of Series-Parallel DAG

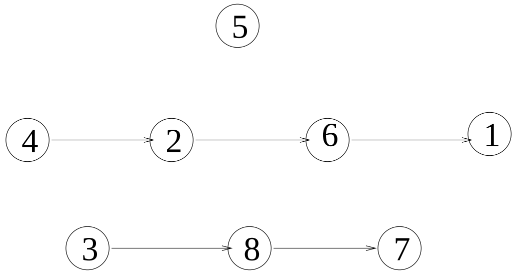


Figure 1: Parallel Chains

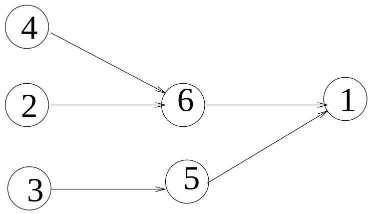


Figure 3: Intree

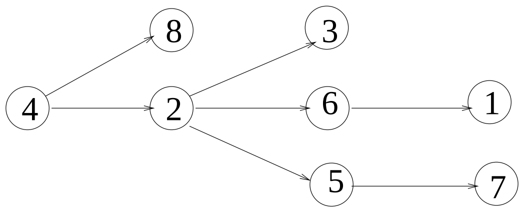


Figure 2: Outtree

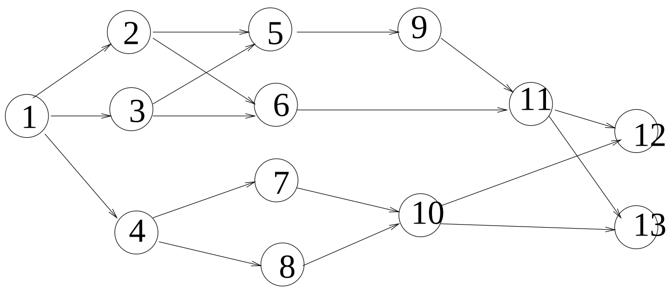


Figure 4: Example of Series-Parallel Digraph

Definition of Series-Parallel Digraph

Definition 2 (Transitive Series Parallel)

(i) A digraph consisting of a single node is a transitive series parallel.

(ii)If $G_1 = (N_1, A_1)$ ***and*** $G_2 = (N_2, A_2)$ ***where*** $N_1 \cap N_2 = \emptyset$***, are transitive series parallel, then,***

$$G = G_1 \times G_2 = (N_1 \cup N_2, A_1 \cup A_2 \cup N_1 \times N_2)$$

is also transitive series parallel. G ***is said to be formed by the series composition of*** G_1 ***and*** G_2***.***

(iii))If $G_1 = (N_1, A_1)$ ***and*** $G_2 = (N_2, A_2)$ ***where*** $N_1 \cap N_2 = \emptyset$***, are transitive series parallel, then,***

$$G = G_1 \cup G_2 = (N_1 \cup N_2, A_1 \cup A_2)$$

is also transitive series parallel. G ***is said to be formed by the parallel composition of*** G_1 ***and*** G_2***.***

Definition 3 (Transitive Closure)

A digraph $\bar{G} = (N, \bar{A})$ ***is the*** *transitive closure* ***of a digraph*** $G = (N, A)$ ***if*** $\bar{A} \supseteq A$***, and for every*** $(i, j) \in \bar{A} \backslash A$***, there is a path from node*** i ***to node*** j ***is*** G***.***

Definition 4 (Series-Parallel)

A digraph is said to be *series parallel* ***if its transitive closure is transitive series and parallel.***

Examples of Series-Parallel DAG: chains, outtrees, intrees and their series or parallel compositions.

Decomposition Tree of Series-Parallel Digraph

- **Any series-parallel digraph can be decomposed into series and parallel components. The decomposition has a tree representation.**
- **For a digraph $G = (N, A)$, there exits a decomposition algorithm of complexity $O(|N| + |A|)$.**
- **Scheduling algorithm assumes the decomposition tree is obtained.**

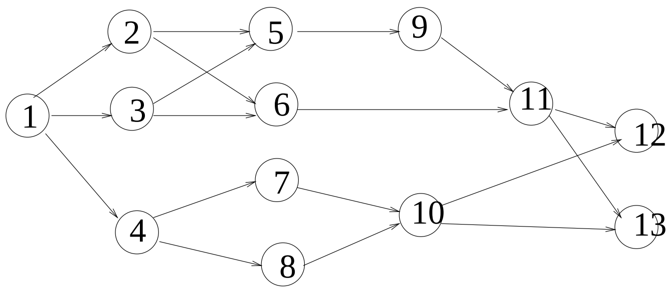


Figure 5: Example of Series-Parallel Digraph

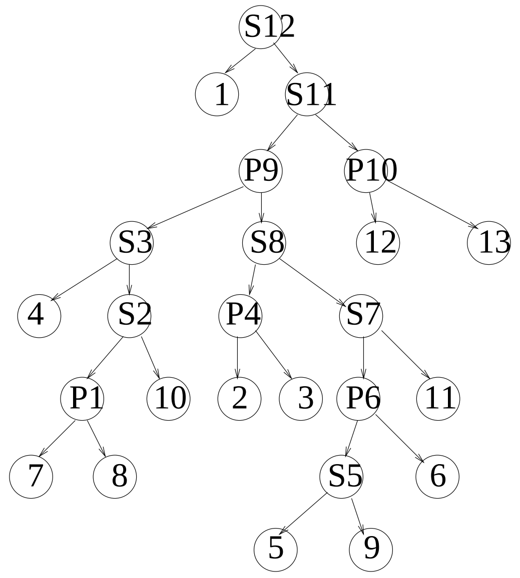


Figure 6: The Corresponding Decomposition Tree

Optimal Schedule for Parallel-Chains

Associate the chain $(i_1, i_2, ..., i_k)$ **with a number** $\xi(i_1, i_2, ...i_k)$**, defined by,**

$$\xi(i_1, i_2, ...i_k) = \min_{1 \le l \le k} \frac{\sum_{j=1}^{l} \alpha_j}{\sum_{j=1}^{l} p_j} \tag{1}$$

Let l^* **be the index for which the above minimum is achieved. We say that job** i_{l^*} *determines* **the chain.**

Algorithm 1 (Sidney's Parallel Chains)

Step 1. ***Transmit the chain with the smallest*** ξ ***factor without interruption upto and including the object that determines the chain.***

Step 2. ***Then repeat Step 1. with the rest of the objects.***

Sydney's Theory

Definition 5 *A nonempty subset $M \subseteq N$ is a (job) module if, for each job, $j \in N \backslash M$, exactly one of the following three conditions holds:*
(i) j must proceed every job in M;
(ii) j must follow every job in M;
(iii) j is not constrained with respect to any job in M.

Theorem 10 *Let M be a module of $G = (N, A)$ and σ is an optimal sequence for M. Then there exists an optimal sequence for N consistent with σ in the sense that the jobs in M appear in the same order as in σ.*

- A singleton subset of N is a job module.
- A chain and a subsequence of consecutive elements in a chain are job modules.
- Given a series parallel G, the subtree rooted at any node of its decomposition tree is identified with an module.
- The theorem implies that elements of a module can be considered to be subject to precedence constraints among themselves in the form of a chain.

Optimal Schedule for Series-Parallel DAG

Algorithm 2 (Sidney's Series Parallel)

Step 1. ***Let*** k ***be the second level from the bottom of the decomposition tree for*** G***.***

Step 2. ***At level*** k***, if a node is labeled*** S***, find a subsequence by simply concatenating its left child with the right child in that order. If a node is labeled*** P***, find a subsequence by running the parallel-chain algorithm on its children. Record the newly found subsequence at the current node.***

Step 3. ***If*** $k > 0$***, then let*** $k = k - 1$***. Repeat Step 2.***

- Lawler improved Sidney's algorithm and found one with complexity $O(n \log n)$.

Optimal Schedule for Random Processing Times

Definition 6 *In a* non-preemptive static policy*, the complete schedule is determined at time 0, and will not change for the entire processing session of the n objects. Non-preemption has the usual meaning.*

Definition 7 *In a* non-preemptive dynamic policy*, every time the server is free, the next object to be transmitted is determined using all the currently available information. Non-preemption has the usual meaning.*

Theorem 11 *The sequencing rule that follows the increasing order of α_i/EX_i maximizes the expected value of the linear utility function in the class of non-preemptive static policies and in the class of non-preemptive dynamic policies.*

- Sidney's Parallel-Chain algorithm still applies with EX_i replacing the fixed processing time, p_i.
- If $s_p/s_q = EX_p/EX_q$, it is enough to order α_i/s_i in increasing order. The condition is satisfied when the processing times for each packet are iid.

A Simple Scheme for Web Application

- **Difficulties / requirements of web object transmission**
  - Not easy to specify due dates and utility functions
  - Due to "on-line" nature, low complexity algorithm is needed
  - The range of processing times can be wide: 5 orders of magnitude
- **Advantages of linear utility function and WSPT schedule**
  - There is only one variable to specify for each object: α_i.
  - Implementation can be hidden from the users, which spares them from making tough choices.
  - It is simple under precedence constraints and randomness in processing time.
  - Given the same weights, WSPT has the features of SPT. In particular, it transmits the smaller objects first. These smaller objects can be: text, metadata portion of a larger object, the basic layer of multilayer-encoded objects, etc.

Algorithm 3 (Object Transmission)

User assigns preference to each object, i, $-\alpha_i = k$, where $k \in \{1, 2, 3, 4, 5\}$. The sender transmits objects in the decreasing order of $10^{-\alpha_i}/s_i$.

Open Issues and Future Directions

- System-level issues are not addressed: application and protocol
  - how to communicate the user preference
  - where to do the scheduling: application or transport
  - who will do the scheduling: sender or receiver
  - granularity of objects
- How else to specify the user preferences: learning-based ideas
- Beyond web applications is the general issue of scheduling of multiple transactions/flows at the end-system
- Unresolved issues in scheduling
  - scheduling for mixed utility functions: step and linear functions
  - mixed scheduling: object-based scheduling and stream-based; bandwidth allocation is a relevant quantity for the latter
  - optimal control formulation of scheduling, which is more natural for preemptive schedules