# Modelling Resource Utilization in Pipelined Query Execution

Myra Spiliopoulou[1] and Johann Christoph Freytag[2]

[1] Institut für Wirtschaftsinformatik
Humboldt-Universität zu Berlin
Email: myra@wiwi.hu-berlin.de
[2] Institut für Informatik
Humboldt-Universität zu Berlin
Email: freytag@dbis.informatik.hu-berlin.de

**Abstract.** Database parallelism offers the potential of efficient query execution, but increases the complexity of optimization, as the impact of the workload of processors and network on competing and communicating processes must be considered. We introduce a query execution model that incorporates the effects of the system workload on pipelined query execution. Our model is general enough to cover bushy and pipelined parallelism for shared-nothing and shared-disk architectures.

## 1 Introduction

One of the most difficult issues in database parallelism is the efficient exploitation of pipelining, because it "does not easily lend itself to load balancing" [3]. Many works on pipelining assume that there are no startup delays between pipeline participants. The effects of latency are simulated and extensively discussed in [6]. Pipelines with latency, termed "e-steady pipelines", are modelled in [1], where the execution time of a pipeline participant is observed as a function of the number of processors assigned to it. However, for large join queries, it is rather expected that multiple processes compete for the same processor.

Pipelined and bushy parallelism [3] is studied in [1, 4, 2, 6], where the communication across *logical* links between interacting processes is also considered. However, the network configuration affects the selection of the *physical* channels to connect two communicating processors. A router may even dynamically choose different routes between the same processors during query execution. Hence, the load of each channel varies over time.

We propose a model for parallel execution, which alleviates those shortcomings. First, we consider pipelines with latency. Second, we incorporate processor and network load into the cost of bushy schedules. Third, we model the varying load on each physical channel during query execution and calculate its impact on communication cost. By incorporating more parameters of the optimizer's search space into the cost function, the approximation of the actual execution time of a schedule becomes more realistic. The parallel architectures we support may conform to the shared-nothing and shared-disk paradigms.

## 2 Modelling the Query Execution Problem

*QEPs and Schedules.* A query execution plan (QEP) is a query tree whose nodes are operators annotated with execution algorithm and data access information. The children of a node $x$ are its "producers"; $x$ is the "consumer". A producer is "blocking" if it must complete execution before its consumer starts, or "pipelining" if each tuple it outputs is immediately consumed [4]. In a bushy QEP, a node may have one pipeline and one blocking producer, e.g. a hash-join, or two pipeline producers, e.g. a merge join on two sorted inputs.

A schedule represents a node-to-processor mapping for the nodes of a QEP. We do not consider node parallelism. Hence, a node is assigned to exactly one processor and corresponds to a "process" in the schedule.

*Execution Model.* Query execution starts by activating the "runnable" processes of the schedule; a process is runnable if it has enough input data to operate upon. The parent of a blocking process becomes runnable when the child completes. The parent of a pipeline process runs in parallel with it, but there is a delay between the start time of the process and the start of its parent [6]. This *latency* is caused by local processing delays on the producer, by transmission delays across the network and by data buffering, if any. A high latency value may justify the replacement of a pipelined edge with a blocking one. Hence, the latency must be known to the optimizer to avoid cosnidering unrealistic QEPs.

Due to blocking and to latency in pipelines, the processes are not active throughout the query execution. Thus, the *system workload varies*, as processor resources and channel bandwidth are dynamically redistributed.

A pipeline executes at the pace of its slowest process. Due to latency, a process affects the execution pace of the pipeline only when it becomes runnable. Therefore, the execution rates of pipelined processes must be synchronized on the fly. We present such a dynamic synchronization mechanism in [5]: We compute the relative execution rates of all producers of a node $x$, and identify the slowest producer $y_1$. We specify two adjustment functions taking values in the $(0, 1]$ range: $adjustConsumer(x, y_1)$ slows down the consumer, while $adjustProducer(x, y_i)$ reduces the execution speed of producer $y_i$.

*The Cost Computation Problem:* given a schedule $S$ and a system configuration, compute $Cost(S)$ as the total elapsed time from the beginning of the first runnable process to the completion of all processes.

$Cost(S)$ is the sum of the execution time of the root process and the elapsed time until the root starts. The latter is computed recursively as the elapsed time until a process starts and its execution time until it has produced enough output for its own consumer to start. A consumer starts when its slowest producer has output enough data. This definition covers blocking and pipelining producers.

Our cost model simulates the execution of the schedule monitoring the evolution of system load. We first consider the impact of latency and multitasking on the cost of a process in a pipeline. In section 4, we study their effects on the whole schedule. The impact of network load is discussed in section 5.

## 3 Process Cost in a Pipeline

### 3.1 Cost of an Individual Process

Let $C(\cdot)$ be the cost of a process executed by a specific algorithm. It includes CPU time and local I/O, but neither remote I/O nor network transfers, which are computed by our model. Cost functions for $C(\cdot)$ are presented in [3].

Let $x$ be a process and $S_x$ the set of all processes running simultaneously on that processor. The processor distributes its resources among $n = card(S)$ processes, where $card(\cdot)$ denotes cardinality. So, the cost of $x$ is $C(x, S_x)$. When the content of $S_x$ changes to $S'_x$, it holds that:

$$C(x,S'_x) = C(x,S_x) \cdot f(S_x,S'_x) \ where \ f(S_x,S'_x) = \frac{\sum_{y \in S'_x} C(y,S'_x)}{\sum_{y \in S_x} C(y,S_x)} \qquad (1)$$

As shown in the next section, recalculation occurs as soon as one process is added or removed from the processor's load, i.e. $card(S'_x) = card(S_x) \pm 1$. Further, the classic relational operators have similar (low) CPU demand and high I/O demand per time unit. Hence, we can assume that their cost increases linearly with the number of processes sharing the processor, so that:

$$f(S_x,S'_x) \approx \frac{card(S'_x)}{card(S_x)}$$

This assumption does not hold for the allocation of memory resources, a problem we intend to address. It will cause the replacement of the above formula by a more complex function. Our results are not affected otherwise.

### 3.2 Cost of a Process in a Pipe

Let $x$ be a process in the schedule, and let $C^{init}(x, S_x)$ be its "initialization time", i.e. the time it needs to produce the data needed by its consumer to start. If $x$ is blocking, $C^{init}(x, S_x) = C(x, S_x)$. For pipeline processes, $C^{init}$ reflects the impact of latency. For the root, which has no consumer, $C^{init}$ is equal to zero.

Let $T^{init}(x, S_x)$ be the elapsed time from query start to the end of the time span $C^{init}(x, S_x)$. Let $y$ be its slowest producer of $x$. Then:

$$T^{init}(x,S_x) = \begin{cases} C^{init}(x,S_x) & , x \text{ is a leaf} \\ T^{init}(y,S_y) + adjustConsumer(x,y) \cdot C^{init}(x,S_x) & , \text{otherwise} \end{cases} \qquad (2)$$

If $x$ is too fast for its producers, it is slowed down by the adjustment function $adjustConsumer(\cdot)$ [5].

Let $C^{end}(x, S_x)$ be the "completion time" of $x$, i.e. its remaining execution time after $C^{init}(x, S_x)$. If the consumer of $x$ runs on the same processor as $x$, then the contents of $S_x$ change into $S'_x$ and the processor's resources must be redistributed. According to Eq.1:

$$C^{end}(x,S'_x) = \begin{cases} C(x,S_x) - C^{init}(x,S_x) & S'_x = S_x \\ C(x,S_x) \cdot f(S_x,S'_x) - C^{init}(x,S_x) & S'_x \neq S_x \end{cases}$$

Let $T^{end}(x, S'_x)$ be the elapsed time from the beginning of query execution to the completion of $x$:

$$T^{end}(x, S'_x) = adjustProducer(z, x) \cdot C^{end}(x, S'_x) + T^{init}(x, S_x) \quad (3)$$

where the adjustment function $adjustProducer(\cdot)$ slows down $x$ if it is faster than its consumer $z$; otherwise it is equal to 1 [5].

In Fig. 1, we show a QEP scheduled on one processor. Its cost is the execution time of $x$ and the elapsed time until it starts execution. This elapsed time is equal to the initialization time of $y_1$, assuming that $y_1$ is slower than $y_2$.

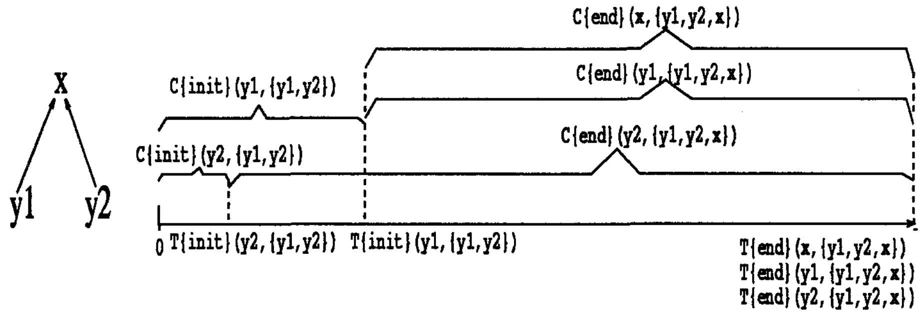

**Fig. 1.** Execution of a pipeline with two pipelining producers

In order to keep the example simple, we have assumed that the three processes finish together. However, there will usually be a delay between the completion of $y_1$ and $y_2$ and that of $x$, implying a further redistribution of resources, as modelled in the next section.

## 4 A Mechanism for Event Capturing

We define as "interesting event" or simply "event" a change on resource utilization, occuring at some point on the processor's time axis. Resource utilization changes when a process starts execution ("Start Event" SE), when it has produced enough data for its consumer to start and must transmit them to it ("initialization Event" IE) and when it terminates ("termination event" TE). The SE for a process coincides with the IE of its slowest producer.

The functions $T^{init}(\cdot)$ and $T^{end}(\cdot)$ defined in Eq.2 and Eq.3 of subsection3.2 mark the occurence of the IE and the TE respectively. We use hereafter the terms "event" and "time point at which the event occurs" interchangeably.

The load on a processor depends on the processes running on it. The load on a network channel depends on the processes using it for data transfer. By the above definition, utilization of these resources is constant between any two adjacent events. Thus, we measure the cost of a schedule by detecting the three aforementioned types of events and placing them on the time axis. This gives us a precise estimation of the execution time on each processor.

## 4.1 Events on One Processor

Let $S^i$ denote the set of processes assigned to processor $i$. The processor is active until all these processes have finished execution, i.e. for the time span $[0, t^i_{end}]$, where $t^i_{end} = \max_{x \in S^i}(T^{end}(x, S^i))$. This time span is divided into intervals by a series of points $0, t^i_1, t^i_2, \ldots, t^i_{end}$, each one corresponding to some event.

Let $A^i$ be the set of processes, for which the IE has not occured yet, and let $B^i$ be the set of processes running to completion after their IEs. When the IEs of all producers of a process have occured, the process becomes runnable and is added to set $A^i$. When its IE occurs, the process migrates to set $B^i$. When its TE occurs, it is eliminated. Processor $i$ completes execution when $A^i$ and $B^i$ become empty. We denote the instance of $A^i$ ($B^i$) at $t^i_j$ as $A^i_j$ ($B^i_j$).

*The $j^{th}$ event.* We want to estimate the time $t^i_j$ at which the $j^{th}$ event occurs. $t^i_j$ corresponds to the earliest among the IEs of the processes in $A^i_j$ and the TEs of the processes in $B^i_j$:

$$t^i_j = \min\left(\min_{x \in A^i_j}(T^{init}(x, A^i_j \cup B^i_j)), \min_{x \in B^i_j}(T^{end}(x, A^i_j \cup B^i_j))\right) \tag{4}$$

Let $x_j$ be the process corresponding to this minimum. Then:

1. If $x_j$ belonged to $A^i_j$, the event indicates that $x_j$ has completed its initialization phase and must migrate from $A^i$ to $B^i$: $A^i_{j+1} = A^i_j - \{x_j\}$ and $B^i_{j+1} = B^i_j \bigcup \{x_j\}$.
   If the consumer of $x_j$, say $w$, is now runnable, i.e. if the IEs of the siblings of $x_j$ have already occurred, then $w$ is added to $A^i_{j+1}$.
2. If $x_j$ belonged to $B^i_j$, the event indicates that its execution is now completed. Then, it must be removed from $B^i$: $B^i_{j+1} = B^i_j - \{x_j\}$.

*Impact of multitasking.* Let $x$ be a process running after the event $t^i_j$. Then:

1. If $t^i_j$ is the SE of $x$, the initialization time of $x$ is $C^{init}(x, A^i_{j+1} \cup B^i_{j+1})$.
2. If the IE of $x$ has not occured before $t^i_j$, then its remaining initialization time is $T^{init}(x, A^i_j \cup B^i_j) - t^i_j$. This time must be adjusted according to Eq.1, into:

$$\underbrace{T^{init}(x, A^i_{j+1} \cup B^i_{j+1}) - t^i_j}_{remaining\ initialization\ time} = (T^{init}(x, A^i_j \cup B^i_j) - t^i_j) \cdot f(A^i_{j+1} \cup B^i_{j+1}, A^i_j \cup B^i_j) \tag{5}$$

   For the computation of $t^i_{j+1}$, we compute the time point of the IE of $x$ for the new resource allocation:

$$T^{init}(x, A^i_{j+1} \cup B^i_{j+1}) = (T^{init}(x, A^i_j \cup B^i_j) - t^i_j) \cdot f(A^i_{j+1} \cup B^i_{j+1}, A^i_j \cup B^i_j) + t^i_j$$

3. If $t^i_j$ is the IE of $x$, the completion time of $x$ is $C^{end}(x, A^i_{j+1} \cup B^i_{j+1})$.

4. If the IE of $x$ has occured before $t^i_j$, the remaining completion time of $x$ is $T^{end}(x, A^i_j \cup B^i_j) - t^i_j$. This time must be adjusted into:

$$\underbrace{T^{end}(x, A^i_{j+1} \cup B^i_{j+1}) - t^i_j}_{\textit{remaining completion time}} = (T^{end}(x, A^i_j \cup B^i_j) - t^i_j) \cdot f(A^i_{j+1} \cup B^i_{j+1}, A^i_j \cup B^i_j) \tag{6}$$

Similarly to the second case above, the TE for $x$ under the new resource allocation will occur at:

$$T^{end}(x, A^i_{j+1} \cup B^i_{j+1}) = (T^{end}(x, A^i_j \cup B^i_j) - t^i_j) \cdot f(A^i_{j+1} \cup B^i_{j+1}, A^i_j \cup B^i_j) + t^i_j$$

## 4.2 Interleaving Events on Different Processors

If a process is assigned to another processor than its consumer, then the producer's IE must be "projected" onto the time axis of the other processor, because data must be transferred from the producer to the consumer. This data transfer affects the network load. We initially describe this projection mechanism without considering the delay caused by data transfer. In the next section, we generalize our formulae to incorporate this delay.

Each processor has its individual clock. The interference of process execution, though, forces us to place the events occuring on different processors in total order on a common time axis. Therefore, we introduce therefore a virtual "reference time axis" on which we place the events of all $p$ processors. The execution time span for the query on the reference axis is $[0, t_{end}]$, where $t_{end} = \max_{i=1\ldots p}(t^i_{end})$. At point 0 (beginning of time axis), we assume without loss of generality, that all processors start processing the query.

We compute the time $t_j$ of the $j^{th}$ event on the reference time axis, by identifying the earliest event that is not already placed on this axis.

$$t_j = \min_{i=1,\ldots,p}(t^i_{j_i}) \tag{7}$$

where we denote as $t^i_{j_i}$ the time of the earliest event on processor $i$, which is not already placed on the reference time axis. We use the double index $j_i$, because the event counters on the different processors are not advanced at the same pace.

Let $t^k_{j_k}$ be the event corresponding to $t_j$. If $t^k_{j_k}$ is the IE of a process whose consumer is located on a different processor $l$, then $t^l_{j_l}$ becomes the time of this IE, i.e. it becomes equal to $t_j$, and the counter $j_l$ is advanced. If this IE caused a process to start, the time point of the next event on $l$ must be recomputed according to Eq.5 and Eq.6 replacing the local time point $t^i_j$ by $t_j$. Otherwise, the IE only needs to be renumbered, as the event counter has been advanced.

*Example.* In Fig. 2, we show a QEP and schedule on two processors $P1$ and $P2$. Each processor has its own time axis, and the events on it are projected on the reference axis. We denote by $x1\_init$ the event occuring at $T^{init}(x_1, \cdot)$ and similarly for the other events. $t\{i\}j$ stands for $t^i_j$.

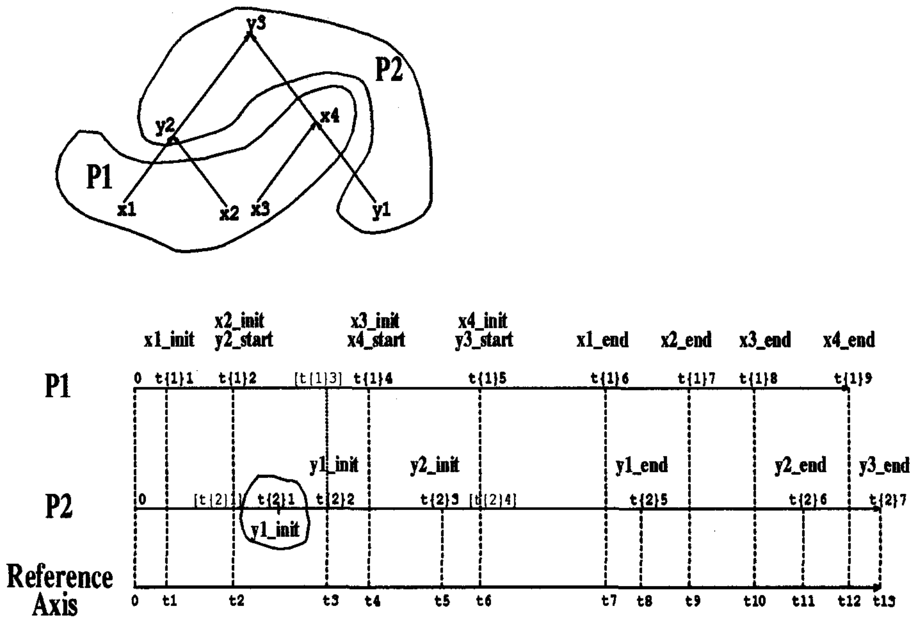
y3
P2
y2
x4
P1
x1
x2
x3
y1
x1_init
x2_init y2_start
x3_init x4_start
x4_init y3_start
x1_end
x2_end
x3_end
x4_end
P1
0 t{1}1 t{1}2 [t{1}3] t{1}4 t{1}5 t{1}6 t{1}7 t{1}8 t{1}9
y1_init
y2_init
y1_end
y2_end
y3_end
P2
0 [t{2}1] t{2}1 t{2}2 t{2}3 [t{2}4] t{2}5 t{2}6 t{2}7
y1_init
Reference Axis
0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13

**Fig. 2.** Execution of a bushy QEP on two processors

Initially, processes $x_1$,$x_2$,$x_3$ run on $P1$ and $y_1$ runs on $P2$. The first event occurs at $t\{1\}1$, so that $t1$ is set equal to $t\{1\}1$. The second event occurs at $t\{1\}2$. It is projected on the time axis of $P2$, as $[t\{2\}1]$, where the square brackets denote the projection operation. This event is also the SE of $y_2$. The original event that occured at $t\{2\}1$ is placed in a circle, because it was computed without considering $y_2$: it is renumbered into $t\{2\}2$ and its value is recomputed.

Event $t3$ corresponds to $t\{2\}2$. Since $y_1$ is one of the producers of $x_4$, $t3$ is projected on the time axis of processor $P1$. This event causes the renumbering of subsequent events on $P1$ but no recomputation because $x_4$ cannot start yet. Subsequent events are computed similarly. TEs are never projected, because they only affect the resource usage of the processor on which they occur.

## 5 Data Transfer Cost

We model the network delay between the time an IE occurs and the time it is perceived by the processor of the consumer.

### 5.1 Channel Load due to Dataflow

We represent the network configuration as a graph $G(V, E)$, where $V$ is the set of processors and $E$ the set of channels connecting them. A schedule assumes a "conceptual" topology of links among the processors that must interact. These links are materialized on the network's channels by a router. We assume that:

- The router may change the mapping of links dynamically during execution.
- The routing criterion is known to the optimizer.
- The path of channels materializing a link does not change between two consecutive events on the reference time axis.

Let $Path_j(i,k)$ be the set of channels selected by the router to materialize the link between processors $i,k$ at $t_j$; this path will not change prior to $t_{j+1}$. Further, let $L^e$ be the set of processes communicating across channel $e \in E$ with their consumers; $L_j^e$ is its instance at $t_j$. The sets $L^e$ $(e \in E)$ reflect the network load, similarly to the sets $A^i$ and $B^i$ $(i \in V)$ reflecting processor load.

We denote by $t_{comm}(e)$ the "ideal transfer time" on a channel $e$, defined as the time required to transfer a data unit across $e$, when $e$ has no other data to transfer. The "actual transfer time" across $e$ at $t_j$ is $card(L_j^e) \cdot t_{comm}(e)$.

Let $block(x^i, x^k)$ be the amount of data units sent at each transmission from $x^i$ to its consumer $x^k$. We assume that those data are accumulated on processor $k$, so that processor $i$ does not fill its local storage with foreign data. If $t_j$ is the IE of $x^i$, then an initial transfer of $block(x^i, x^k)$ data units must be performed from processor $i$ to $k$. Its cost is:

$$transferCost(x^i, x^k, t_j) = block(x^i, x^k) \cdot \sum_{e \in Path_j(i,k)} card(L_j^e) \cdot t_{comm}(e) \quad (8)$$

In order to project the IE of $x^i$ onto the time axis of $k$, this transfer delay must be taken into account:

$$t_{j_k}^k = t_j + transferCost(x^i, x^k, t_j) \quad (9)$$

## 5.2 Remote I/O Cost

The network load is further affected by remote I/O. If each processor has its own local storage, as in the shared-nothing architecture, then intermediate results are stored locally and remote I/O may only occur at leaf nodes. We model this case by introducing "I/O processes", dedicated to retrieve data from local discs and to transmit them to the network. As this transmission can take place in pipeline mode, remote I/O cost is a special case of dataflow cost.

If some processors have no access to local storage, as in the shared-disk architecture, all their I/O is remote. Remote I/O for data transmission can be modelled by the I/O processes described above. Remote I/O on the temporary data of joins must be modelled differently, because it can occur at any time, even between adjacent events. An initial solution to this problem is described in [5].

## 6 Conclusions

We have presented a model incorporating resource utilization in parallel query execution. Our model covers the exploitation of bushy parallelism and pipelining in shared-nothing and shared-disk architectures. We consider a complex pipeline scheme, in which we integrate the impact of latency in pipe execution and support the dynamic synchronization of processes in a pipe.

In order to provide the optimizer with a realistic estimate of schedule cost, we compute the system load by simulating schedule execution on each processor involved. This simulation mechanism is based on the detection of interesting events, namely events that signal a change in the system load, and on their placement on a time axis. The time span covered by those events is the total duration of schedule execution.

Our model is more general than typical scheduling models, but also more vulnerable to wrong estimations concerning system load, especially for mixed loads. Schedule cost computation increases quadratically to the query size, making our model too expensive for exhaustive optimizers. However, our envisaged application area is that of parallel querying for decision support: The query sizes make the usage of non-exhaustive strategies imperative. Query processing cost is higher than optimization cost by orders of magnitude. The system load implied by one query makes the launching of multiple simultaneous queries prohibitive.

The second application area of our model is the analysis of the solution spaces for parallel schedules. Its generality makes it suitable as a reference model to study the behaviour of simpler and faster models. In particular, it is appropriate for a detailed analysis of the shape of the search space of parallel schedules, so that the proximity of the search spaces of other cost models can be qualitatively evaluated. Our immediate plans include such a detailed analysis for bushy trees of large join queries and a comparative study of simpler cost models.

## References

1. Sumit Ganguly, Apostolos Gerasoulis, and Weining Wang. Partitioning pipelines with communication costs. In *Int. Conf. on Information Systems and Management of Data*, pages 302–320, Bombay, India, 1995.
2. Minos Garofalakis and Yannis Ioannidis. Multi-dimensional resource scheduling for parallel queries. In *SIGMOD Int. Conf. on Management of Data*. ACM, 1996.
3. Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.
4. Waqar Hasan and Rajeev Motwani. Optimization algorithms for exploiting the parallelism-communication tradeoff in pipelined parallelism. In *Int. Conf. on Very Large Databases*, pages 36–47, Santiago, Chile, 1994.
5. Myra Spiliopoulou and Johann Christoph Freytag. Modelling the dynamic evolution of system workload during pipelined query execution. Technical Report ISS-20, Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, Germany, 1995. http://www.wiwi.hu-berlin.de/institute/iwi/info/research/iss//papers/ISS20.ps.
6. Anita N. Wilschut and Peter M.G. Apers. Dataflow query execution in a parallel main-memory environment. In *First Int. Conf. on Parallel and Distributed Information Systems*, pages 68–77. IEEE, 1991.