データベースシステム　122－13
(2000. 7. 26)

# コンティニュアス・データベースを対象とする最適コンピュータ資源割り当て

陳　幸、　清木　康

神奈川工科大学　情報工学科
〒243-0292　神奈川県厚木市下荻野 1030
神奈川工科大学　情報工学科
TEL:046-291-3063(直通)　FAX:046-242-8490(事務室)
chen@ic.kanagawa-it.ac.jp

キーワード　データベース、資源割り当て、マルチメディア・データ処理

# Optimal Computer Resource Allocation for Continues Media Database

Xing Chen and Yasushi Kiyoki
Kanagawa Institute of Technology
Department of Information & Computer Sciences
Kanagawa Institute of Technology
1030 Simo-Ogino, Atsugi-shi
Kanagawa, 243-0292 Japan
Tel: +81-46-291-3063
Fax: +81-46-242-8490
E-mail: chen@ic.kanagawa-it.ac.jp

Abstract

In this paper, we apply a stream-oriented database processing scheme for the continues media data processing. We focus on the presentation operations for the continues media data. In this kind of processing, database operations are repeatedly executed. For example, in the case that several music data streams are presented simultaneously, as that, stream-A and stream-B are presented as the background music, stream-C is presented as the main melody music. It is clear that when the background music is repeated, the database operation for processing the background music must be repeatedly executed. It is shown in this paper that the overhead of the re-execution can be reduced by optimal memory resources allocation. Therefore, we introduce an optimal memory allocation method. A heuristics is utilized in the method. Based on the heuristics, the calculation complexity of the method is reduced. We also discuss the property of the method and show several calculation examples to clarify the efficiency of the method.

key words　database, resource allocation multimedia data processing

# Optimal Computer Resource Allocation for Continues Media Database

Xing Chen
Department of Information &
Computer Sciences
Kanagawa Institute of Technology
Atsugi, Kanagawa 243-0292 Japan
chen@hasu.cc.tsukuba.ac.jp

Yasushi Kiyoki
Faculty of Environmental
Information
Keio University
Fujisawa, Kanagawa 252, Japan
kiyoki@sfc.keio.ac.jp

## Abstract

In this paper, we apply a stream-oriented database processing scheme for the continues media data processing. We focus on the presentation operations for the continues media data. In this kind of processing, database operations are repeatedly executed. For example, in the case that several music data streams are presented simultaneously, as that, stream-A and stream-B are presented as the background music, stream-C is presented as the main melody music. It is clear that when the background music is repeated, the database operation for processing the background music must be repeatedly executed. It is shown in this paper that the overhead of the re-execution can be reduced by optimal memory resources allocation. Therefore, we introduce an optimal memory allocation method. A heuristics is utilized in the method. Based on the heuristics, the calculation complexity of the method is reduced. We also discuss the property of the method and show several calculation examples to clarify the efficiency of the method.

## 1. Introduction

As the rapidly decreasing costs of storage and advancements in compression techniques, it is easy to store continues media data such as video and audio data in database systems. It is possible to store thousands hours of video data in a database system. Continues media data operations, such as data retrieving and presentation must provided to retrieve and view continues media data. We have presented methods for continues data retrieval and presentation [1,2].

The simplest way of presenting a continues media data stream is to play the stream from the beginning to the end. In a continues media database system, it is necessary to provide more efficient methods to presenting the continues media data stream. That is, to play several continues media data streams in the order according to queries. For example, when a stream, stream-A, is played, another stream, stream-B, will interrupt it. Then stream-B is played. When the playing of stream-B is finished, stream-A will be played continuously. The streams, stream-A and stream-B will be presented according to a query in the order as: stream-A, stream-B, and stream-A. Furthermore, it is required that stream-A and stream-B are played simultaneously, stream-A is played as the background of stream-B. When more than one continues media data streams are presented in a same time, optimal memory allocation is required because there are no enough memory resources to store several continues media data streams in a same time.

Memory allocation is important in database systems. Many memory allocation methods are proposed for the relational database system [3,4]. Some new memory allocation algorithms are proposed for advanced database systems [5,6].

We have proposed a stream-oriented database processing scheme for the advanced database processing [7,8]. This scheme is based on the stream processing principle [9]. The functional programming concept [10] and the demand-driven evaluation method [11] are used in the scheme.

The stream-oriented scheme is suitable for the continues media data processing according to our research.. In this paper, we will introduce a method developed form [12] to improve the processing efficiency of continues media data.

## 2. The stream-oriented database processing scheme

In the stream-oriented database-processing scheme, a stream is an ordered sequence of data which are arranged according to the order of production. Each element of a stream corresponds to a page, therefore, the ordered sequence of pages is manipulated as a stream. In the scheme, the demand-driven evaluation [11] is used for processing database operations. Since the execution and suspension of the computation are controlled by demands, it is possible in a natural way to control the computation. The computation of the operation producing a stream can be controlled, according to the computation speed of the operation consuming the stream. Therefore, the demand-driven evaluation shows outstandingly effective advantages in the following aspects: which gives better control of parallelism, a more selective evaluation, and a natural way of handling large amounts of data within limited resource environments.

In the stream-oriented scheme, a function produces its return value as a stream, and an other function consumes the stream as the actual argument. The function which produces a stream is referred as producer function, and the function which consumes the stream is referred as the consumer function.

In the demand-driven evaluation, when an argument of a function is encountered during the execution of the function body, the argument is evaluated by issuing a demand to its producer-function.

When more than two streams have to be executed simultaneously, the producer function must begin the computation before the consumer function needs the actual argument. In our scheme, a demand is pre-issued to the producer function before the consumer function refers to the formal argument. The consumer function of the stream-type argument propagates a first demand to the producer function before the argument is encountered. As a result, the producer function is activated in advance and begins to produce the first grain of stream elements. The producer function does not create every stream element for a single demand. The function creates only a single grain of stream elements for a single demand. After the producer function completes the production of a grain, it suspends the computation until a subsequent demand arrives.

In the scheme, the "call-by-name" argument evaluation method [10] is applied in the demand-driven evaluation. In call-by-name, the formal argument is re-evaluated whenever the argument is encountered in the function body. In this evaluation, the actual argument is removed after a reference to it is completed. When the argument is needed to be re-evaluated, the operations for producing the argument have to be re-executed.

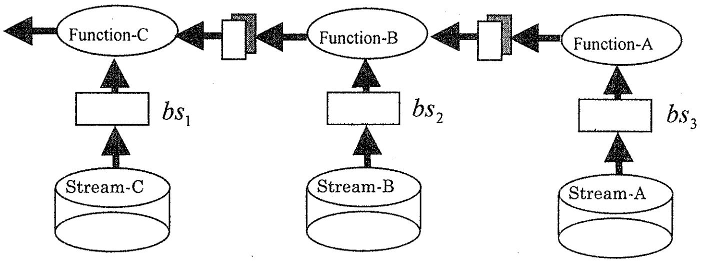


Fig. 1 Stream-oriented processing

## 3. The relationship between memory resources and the processing of continues media data

In the case that several continues media data streams are processed simultaneously, for example, music data streams, stream-A and stream-B are presented as the background music, stream-C is presented as the main melody music. The stream-oriented scheme is efficient to support the processing as shown in Fig. 1. In the stream-oriented scheme, each operation is decomposed into a function tree. Nodes of the tree represent functions and arrows represent streams. In Fig. 1, stream-A is consumed by function-A, stream-B is consumed by function-B and stream-C is consumed by function-C. The produced stream of function-A is consumed by function-B and the produced stream of function-B is consumed by function-C.

In Fig. 1, if it is supposed that the background music stream-A and stream-B are repeated during the playing of the main melody stream-C, function-A and function-B are repeatedly executed.

The processing efficiency of the repeated execution of functions is affected by memory resources. Let's use Fig. 1 as an example to explain the relationship between the number of repeated execution of functions and memory resources.

Functions consume the streams by caching them in buffers. In Fig. 1, three buffers $bs_1$, $bs_2$ and $bs_3$ are utilized to cache the streams, stream-A, stream-B and stream-C.

If all the stream elements are cached in the buffers, it is not necessary to refresh the buffer. If only half of the total stream elements are cached in the buffer $bs_3$, the buffer has to be refreshed twice, and function-C has to be re-executed twice. The number of repeated executions of a function is affected by the buffer size.

## 4. The optimal memory allocation method

The following symbols are used in our discussion.

- $n$ : the total number of buffers,
- $M$ : memory resources,
- $bs_i$ : buffer and its size,
- $opbs_i$ : the optimal buffer size of $bs_i$,
- $d_i$ : the steam size referenced by function-I
- $T_{I/O}$ : the I/O operation time that is needed to read/write stream elements from the I/O device into the buffer of function-$i$,
- $T_{tr-i}$ : the time needed to transmit one byte from I/O device to the buffer of function-$i$,
- $T_{st}$ : the time needed when an I/O access command is sent from function-$i$ until the first byte of the stream is arrived to the buffer of function-$i$,
- $T_{c-i}$ : the stream processing CPU time of function-$i$ (constant),
- $T_Q$ : the processing of the query.

The memory allocation problem is to minimize the database processing time $T_Q$ in the limited memory resources:

$$\min(T_Q(bs_1, bs_2, \cdots, bs_n)), \quad (1)$$

subject to:

$$M = \sum_{i=1}^{n} bs_i . \quad (2)$$

Continues media data can be presented in different ways according queries, even if they are presented in the way as shown in Fig. 1. In the following, we discuss one of the presentation. In the presentation, the stream-A and stream-B must be presented from the beginning to the end for every requirement. The following formula shows the processing time for this kind of processing.

$$\begin{aligned} T_Q &= T_1 \\ T_1 &= T_{c-1} + T_{tr-1} \cdot d_1 + \left\lceil \frac{d_1}{bs_1} \right\rceil \cdot (T_{st} + T_2) \\ &\vdots \\ T_i &= T_{c-i} + T_{tr-i} \cdot d_i + \left\lceil \frac{d_i}{bs_i} \right\rceil \cdot (T_{st} + T_{i+1}) \\ &\vdots \\ T_n &= T_{c-n} + T_{tr-n} \cdot d_n + \left\lceil \frac{d_n}{bs_n} \right\rceil \cdot T_{st} \end{aligned} \quad (3)$$

The exhaustive computation can be used to find the optimal memory allocation given by Formula (1) and (2). The exhaustive computation requires evaluating Formula (3) for all possible allocations of memory space to each buffer. This method needs long calculation time in the case that the memory space $M$ is large. On the other hand, since the computation for optimal allocation often relies on the estimated parameters, the exhaustive calculation may not always give the optimal allocation result.

In this section, we use a heuristics for the optimal memory allocation. The basic idea of our method is shown in the following.

For a query decomposed into $n$ functions, the optimal buffer sizes are calculated as follows;

1. The buffer size $bs_1$ is set as $M/n$.
2. The buffer size $bs_1$ is increased by $x$ as:

   $bs_1 = M/n + x, \quad 0 \le x < M$. The other buffer sizes are set as:

   $$bs_i = \frac{M - bs_1}{n-1}, \quad 2 \le i \le n.$$

   Thus, the processing time $T_1$ is the function of $x$:

   $$T_1 = T_1(x).$$
3. Drive out the value $x_1$ by which the value of $T_1$ is minimal.

When the value $x_1$ is driven out, the optimal buffer size $bs_1$ is obtained:

$$opbs_1 = M / n + x_1 .$$

After the optimal size $bs_1$ is obtained, the optimal buffer size $bs_2$ is calculated. The same two steps of above are used. In general, for calculating the optimal buffer size $opbs_i$, the buffer size $bs_i$ is set as:

$$bs_i = \frac{M - \sum_{k=1}^{i-1} opbs_k}{n-(i-1)} + x , \qquad (4)$$

the other buffer sizes are set as:

$$bs_j = \frac{M - \sum_{k=1}^{i-1} opbs_k - bs_i}{n-i}, \quad i+1 \le j \le n \quad (5)$$

When the value of $x_i$ is driven out by which the value of $T_i$ is minimal, the optimal buffer size $opbs_i$ is obtained as:

$$opbs_i = \frac{M - \sum_{k=1}^{i-1} opbs_k}{n-(i-1)} + x_i \qquad (6)$$

Consider the ceiling function in Formula (3), it is not needed to set a buffer space larger than the space which will cause the same number of the re-executions. Thus, the buffer size is set as:

$$bs_i = \left\lceil \frac{d_i}{\frac{M - \sum_{k=1}^{i-1} opbs_k}{n-(i-1)} + x} \right\rceil . \qquad (7)$$

In our method, the buffer size $bs_i$ is increased by $x$ in order to reduce the value of re-execution factor $\left\lceil \frac{d_i}{(bs_i + x)} \right\rceil$. Consider the ceiling function, the value of $x$ is increased as:

$$x = \left\lceil \frac{d_i}{\left\lceil \frac{d_i}{bs_i} \right\rceil - p} \right\rceil - bs_i . \qquad (8)$$

In the formula, $P$ is an integer number: $P = 1, 2, \cdots$. When $P$ is increased to $P+1$, we say that $bs_i$ is increased in a step. Combing Formula (8) with Formula (4), we obtained Formula (9):

$$bs_i = \left\lceil \frac{d_i}{\left\lceil \frac{d_i}{\frac{M - \sum_{k=1}^{i-1} opbs_k}{n-(i-1)}} \right\rceil - P} \right\rceil . \qquad (9)$$

The maximal value of $P$ is

$$P_{\max} = \left\lceil \frac{d_i}{\frac{M - \sum_{k=1}^{i-1} opbs_k}{n-(i-1)}} \right\rceil - \left\lceil \frac{d_i}{M - \sum_{k=1}^{i-1} opbs_k} \right\rceil . \qquad (10)$$

Summarize all the above considerations, we present our allocation algorithm.

**Step-1:**

Steps 2, 3 and 4 are repeated to calculated the optimal buffer sizes. The initial value of $i$ is 1 and the end value of $i$ is $n-1$; $i$ is increased by 1 after each loop.

**Step-2:**

The buffer size $bs_i$ is calculated by Formula (9) and the buffer size $bs_j$, $i<j<n$, is calculated by Formula (5). The value of $opbs_0$ is 0.

**Step-3:**

The value of $T_i$ in Formula (3) is calculated. The parameters of the buffer sizes for the calculation are those obtained from Step-2. The value of $T_i$ is assigned to the variable $T_{\min}$ as the initial minimal value. The buffer size $bs_i$ is assigned into the variable $opbs_i$ as the initial optimal size.

**Step-4**

The buffer size $bs_i$ is increased in a step by increasing $P$ in Formula (9): $P=P+1$. The steps 2, 3 and 4 are repeated until $P$ is increased to its maximal value

$P_{\max}$.

**Step-5:**

The remained memory space is assigned to the variable $opbs_n$ as the optimal buffer size.

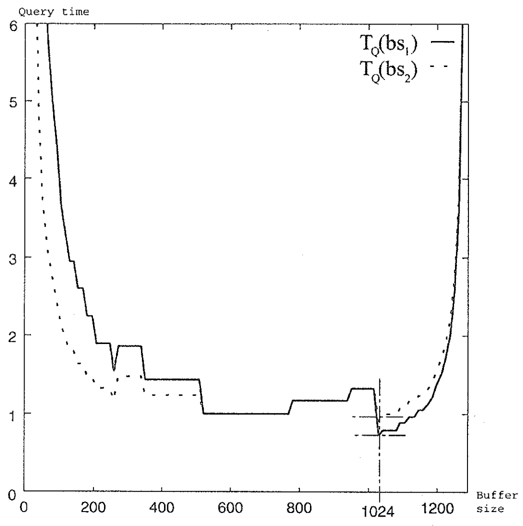
$$opbs_n = M - \sum_{i=1}^{n-1} opbs_i .$$

When the above 5 steps are finished, the optimal buffer sizes, $opbs_1, opbs_2, \cdots, opbs_n$, are obtained.

## 5. Discussion and Calculation Examples

Analyzing Formula (3), it is clear that processing time $T_i$ is longer than that of $T_{i+1}$. If increasing the buffer size $bs_{i+1}$ needs memory resources as that of increasing the buffer size $bs_i$, more benefit on increasing processing performance will be obtained by increasing the buffer size $bs_i$ rather than that by increasing the buffer size $bs_{i+1}$. Fig. 2 shows an calculation example. In Fig. 2, there are three buffers. The figure shows the relationship between processing time $T_Q$ and the buffer size $bs_1$ by fixing the buffer sizes of $bs_2$ and $bs_3$. It also shows the relationship between processing time $T_Q$ and the buffer size $bs_2$ by fixing the buffer sizes of $bs_1$ and $bs_3$. It is clear that in both cases processing times are reduced to minimum. However, the processing time is shorter by increasing buffer size $bs_1$ rather than that by increasing buffer size $bs_2$.

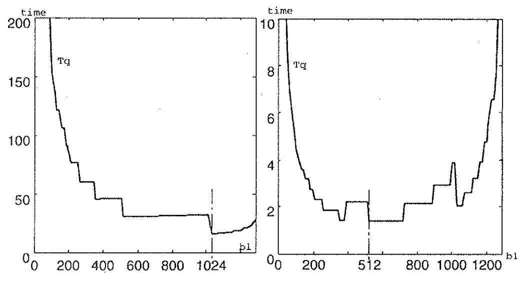
The stream processing CPU time and the I/O operation time also affect the optimal buffer sizes. Fig. 3 shows examples about that. Fig. 3(a) shows the case of CPU sensitive processing. In this case, the optimal buffer size $bs_1$ is 1024 pages. Fig. 3(b) shows the case of I/O sensitive processing. In this case, the optimal buffer size $bs_1$ is 512 pages.



Parameters:

n=3 $T_{C1}$=0.10(second)
M=1800(page) $T_{C2}$=0.10(second)
$d_1$=$d_2$=$d_3$=1024(page) $T_{C3}$=0.0(second)
1 page=1024 bytes $T_{st}$=0.04(second)
$T_{tr}$=5X$10^{-7}$ (second)

Fig. 2: The query processing times



Parameters:

n=3
M=1408(page)
$d_1$=$d_2$=$d_3$=1024(page)
$T_{C1}$=0.10(second)
$T_{C2}$=14.5(second)
$T_{C3}$=0.0(second)
$T_{st}$=0.04(second)
$T_{tr}$=5X$10^{7}$ (second)
1 page=1024 bytes

(a)

Parameters:

n=3
M=1408(page)
$d_1$=$d_2$=$d_3$=1024(page)
$T_{C1}$=0.10(second)
$T_{C2}$=0.10(second)
$T_{C3}$=0.0(second)
$T_{st}$=0.04(second)
$T_{tr}$=5X$10^{7}$ (second)
1 page=1024 bytes

(b)

Fig. 3: CPU and I/O sensitive queries

## 6. Conclusion

In the processing of presentation for continues media data, the overhead of the repeated execution of database operations is one of the main factors affecting the processing performance. The overhead of the repeated execution of database operations can be reduced by optimal memory allocation. In this paper, we applied the stream-oriented scheme to processing the presentation for continues media data. Based on the scheme, a method for the optimal memory resources allocation is presented. In order to simplify the optimal allocation problem of the database operations, we introduced a heuristics. The optimal memory resources allocation method contributes to reducing the overhead of the re-execution of functions and as the result, the processing performance is improved.

We have discussed the property of the method and presented several calculation results. From the results, it is clarified that the query performance is effectively improved by utilizing the method.

## References

[1] Y. Kiyoki, T. Kitagawa, and T. Hayama, "A metadatabase system for semantic image search by a mathematical model of meaning," *Multimedia Data Management 1998*: pp. 191-222.

[2] .Y. Sato and Y. Kiyoki, "A semantic associative search method for media data with story," *Proceedings of the 18th IASTED International Conference on Applied Informatics*, pp., Feb., 2000.

[3] Won Kim. "A new way to compute the product and join of relations," *In Proceedings of the A CM-SIGMOD Conference on Management of Data*, pp. 179-187, 1980.

[4] J. L. Wolf, B. R. Iyer, K. R. Pattipati and J. Turek, "Optimal buffer partitioning for the nested block join algorithm," *In Proceedings of 7th IEEE International Conference on Data Engineering*, pp. 510-519, April 1991.

[5] P. Liu, Y. Kiyoki, and T.Masuda, "A dynamic resource allocation strategy in the stream-oriented parallel processing scheme for relational database operations," *Transactions of Information Processing Society of Japan*, Vol. 29, No. 7, pp. 656-668, 1988.

[6] P. Liu, Y. Kiyoki, and T. Masuda, "Efficient algorithms for resource allocation in distributed and parallel query processing environment," *In Proceedings of 9th IEEE International Conference on Distributed Computing Systems*, pp. 316-323, 1989.

[7] Kiyoki, "A Stream-Oriented Parallel Processing Scheme for Relational Database Processing," *Proc. IEEE International Conference on Parallel Processing*, pp.1013-1020, 1986.

[8] Kiyoki, T. Kurosawa, K. Kato and T. Masuda, "The Software Architecture of a Parallel Processing System for Advanced Database Applications," *Proc. 7th IEEE International Conference on Data Engineering*, pp. 220-229, 1991.

[9] G. Kahn and D. MacQueen, "Coroutines and Networks of Parallel Processes", *Proc. IFIP'77*, pp. 993-998, 1977.

[10] P. Henderson, *Functional Programming: Application and Implementation*, Prentice-Hall, Englewood Cliffs, 1980.

[11] P. C. Treleaven, D. R. Brownbridge and R. P. Hopkins, "Data-driven and Demand-driven Computer Architecture," *ACM Comput. Surv.*, Vol. 14, No. 1, pp. 93-144, 1982.

[12] X. Chen and Y. Kiyoki, "The Optimal Memory Allocation by Utilization of Disk Space for Stream-Oriented Database Processing," *15th IASTED International Conference on Applied Informatics*, pp. 45-50, 1997.