ZESZYTY NAUKOWE POLITECHNIKI BIAŁOSTOCKIEJ 2009
Informatyka – Zeszyt 4

Marta K. Smolińska[1], Zenon A. Sosnowski[1]

PARALLEL FUZZY CLUSTERING FOR LINGUISTIC SUMMARIES

Abstract: The linguistic summaries have the associated truth value so they can be used as predicates. We use summaries of the form "most objects in population *P* are similar to o_i" to find typical values in population *P*. Then typical values are used in fuzzy clustering algorithm. Disadvantage of this algorithm is its complexity. For the purpose of processing the huge number of data, we decided to use parallel computing mechanism to implement this algorithm, and run it on the cluster machine. We use MPI (Message Passing Interface) to communicate between processes, which work on different processors. This paper presents this parallel algorithm and some results of experiments.

Keywords: linguistic summary, fuzzy clustering, parallel computing

1. Introduction

In the preceding article [1] we presented the algorithm of clustering objects in object-oriented database, based on linguistic summaries, according to Yager's approach [2,3]. We also introduced the certain modification of Yager's algorithm which improves the clustering but extremely increases complexity, which is high, in this algorithm without modification. For the purpose of processing the huge number of data, we decided to use parallel computing mechanism to implement this algorithm, and use it on the cluster machine. We use MPI (Message Passing Interface) to communicate between processes, which work on different processors. This paper presents this parallel algorithm and some results of experiments.

The paper is organised as follows. Section 2. describes the linguistic summary. In Section 3 we describe typical values, and in Section 4. their use for searching typical clusters is presented. In Section 5. we introduce parallel version of clustering algorithm. Then in Section 6. we present the results of experiments. Finally, the conclusion will follow in Section 7.

[1] Faculty of Computer Science, Bialystok Technical University, Białystok

2. Linguistic summary

Rasmussen and Yager in [2,3] propose linguistic summary of the form "Q objects in P are S". In this paper we will use notation $Summary(Q,P,S)$ to express that summary. For example linguistic summary can look as follows *"most people are tall" (Summary(most, people, tall)), "few tall people are light" (Summary(few, tall people, light)*.

Q is the quantity in agreement (a linguistic quantifier as most, few etc.), P is a data collection and S is called the summarizer (e.g. young, tall), which very often is a fuzzy predicate.

The truth value $\tau[0,1]$, called the measure of validity is associated with a linguistic summary. It provides an indication of how compatible the linguistic summary with the population is. As a population we mean a fuzzy set of objects. Each object o in a population P has a degree of membership to P - $\mu_P(o)$ (we will also use symbol $o.\mu$).

Measure of validity is calculated as the grade of membership of the proportion of objects in P that satisfy S (eq. 1).

$$\tau = \mu_Q\left(\frac{card_f(S \cap P)}{card_f(P)}\right) \tag{1}$$

$\cap$ is defined as the minimum aggregation. $card_f$ is the fuzzy cardinality of a fuzzy subset and is defined by equation 2

$$card_f(P) = \sum_{o_i \in P} \mu_P(o_i) \tag{2}$$

Complexity of calculating the true degree τ of a linguistic summary is $O(N)$, where N is the number of objects in P.

3. Typicality - typical values

Typicality of an object tells as how much the object is typical in the population. It is calculated by means of linguistic summary, which like a fuzzy predicate has a truth value. So we can use it as a predicate. Value of typicality $t(o_i)$ (we will use also syntax $o_i.t$) of object o_i is the minimum of membership degree of o_i and measure of validity of summary: "most objects in P are similar to o_i" like in eq. 3

$$t(o_i) = o_i.t = min(\mu(o_i), Summary(most, P, \text{Similar\_to}(o_i))) \tag{3}$$

In order to query about typical objects in P, we can establish a new population *typical*, which will consist of objects from P with associated typicality. α-cut can be used to cut off objects with typicality lower than α. Complexity of calculating typicality for one object is $O(N)$, where N is the number of objects in P. So creating population *typical* (calculating all objects in population) has a complexity $O(N^2)$.

4. Fuzzy clustering algorithm

This algorithm is based on Rasmussen approach [2]. Algorithm 1 presents an idea of finding typical clusters using linguistic summaries. In [1] we modified this algorithm by adding step 4. This gives better clustering results but increases complexity from $O(N^2)$ to $O(N^3)$. Practically this complexity depends on the number and size of the obtained clusters.

Algorithm 1 TypicalClusters(P, Q, Sim)

```
Require: $P-$ population, $Q-$ function (pointer to function) of fuzzy quantifier, $Sim(o_i, o_j)$ - similarity
    function $o_i, o_j \in P$
 1: For each object $o$ in population count its typicality $o.t$. Add them all to the population typical,
    remember their typicality $o.t$ and membership degree to the original population $o.\mu$.
 2: Let the most typical object $o'$ $(\forall o \in typical : o'.t \geq o.t)$ be a seed for a new cluster $C_i$.
 3: Find objects $o_j$ from typical that are close to any object of $C_i$, that is $\exists o \in C_i$: $Similarity(o_j, o) \geq$
    $\alpha$. Add them to the cluster $C_i$ and remove from typical. Continue until no more objects are added.
    The cluster $C_i$ will then constitute a typical cluster.
 4: For each object $o$ in typical count its typicality $o.t$, among objects remaining in typical. In this
    calculations we use membership degree that objects had in original population $o.\mu$.
 5: To find next cluster repeat from step 2, as long as there are objects in typical, and the most typical
    object $o'$ has a degree of typicality greater or equal than $\beta$ $(o'.t \geq \beta)$.
 6: return Clusters $C_0 \ldots C_n$ {$n$ - number of created clusters}
```

Clustering Quality

We tested our and Rassmusen algorithms on database formed from pixels of an image. To create the resultant image, we assign colour of the first object in cluster (which was added to cluster as first) to all pixels from this cluster. We run this algorithm with factor $\beta = 0$ to cluster all pixels. To compare results with original image we use Euclidean distance metric between colours in RGB space eq. 4.

$$D(o,r)=\sum_{i,j}\sqrt{(o[i,j].R-r[i,j].R)^2+(o[i,j].G-r[i,j].G)^2+(o[i,j].B-r[i,j].B)^2} \tag{4}$$

where o and r are original and result image respectively. This metric is chosen for its computational speed and simplicity.

Rassmusen's algorithm created 305 clusters and the difference D between result and original image was equal to 13 745 431. Modified version created 301 clusters and D was equal to 5 605 994.5 . There is less colours (less created clusters) in the result image and difference from original is over 2 times less than in algorithm without modification.

5. Parallel Fuzzy clustering algorithm

This parallel algorithm was implemented and tested on populations with the large amount of data. In our parallel implementation we use MPI (Message Passing Interface). We execute our algorithm on many processors. One process on each processor.

We have one master process - "special process designated to manage the pool of available tasks" and many slaves - "processes that depend on the master to obtain work" [4]. Each process is executed on different processor of cluster machine.

In our algorithm there are two situations.

First: when we can divide work among processes and we know that it will take the same time (the same number of computations). This is when we calculate typicality for objects of population. Then we use static mapping of tasks onto processes. Each process calculates the part of objects.

Second situation: when we check if object is similar to cluster. Object is similar to the cluster, if it is similar, to one or more elements of cluster. In optimistic situation, when object is similar to the first object of the cluster, the computation of similarity is done only once. In pessimistic situation - when object is not similar to the cluster, we have to calculate similarity as many as cluster size. To obtain load balancing we use dynamic mapping of tasks to processors. The master tells slaves which object they have to check of similarity to the cluster. When slave process returns result to master, then it receives another task if there are any.

Message in MPI has a tag. We use it to inform other processes what message is sent. In many situations slaves cannot establish what message will be received. So it checks (with MPI_Probe) what tag the message has, and does adequate operations e.g. allocates memory and then receives message.

The idea of our parallel algorithm is as follows. (Algorithms 2 and 3 present in detail master and slave functions. As a function $typicality(o,P)$ we mean $typicality(o,P) = min(\mu(o), Summary(most, P, \text{Similar\_to}(o))))$

In our case each process needs all the objects, because computing typicality of one object requires all other objects. Master packs the population P, and sends it to all other processes - slaves. The slaves receive packed population and unpack it. So each process has a copy of data. Then slaves calculate typicality of the part of the population - each slave different part. Processes in communicator have identifiers from 0 to $p-1$ where p is number of processors. In our case process with identifier 0 becomes master. If N is the population size (number of objects), process with identifier p_{id} considers objects starting from the object with index equal to $\frac{N}{p-1} * (p_{id} - 1)$ ending with index equal to $(\frac{N}{p-1} * p_{id}) - 1$. When N is not divisible by $(p-1)$ then master calculates the rest of objects with indices from $N - (N \bmod (p-1))$ to $N-1$ (indices of the population are zero-based). Then each slave sends results to master, which accumulates them and sends back to all slaves. So each slave can create copy of the population $typical$.

All processes create new cluster containing one object - the most typical. Master saves all clusters in an array. Slaves do not need to keep all clusters. They use only one actually created cluster.

In the next step, the master sends to each slave index of the object, it have to calculate, if that object is similar to the cluster, or not. Then slave sends back the information, and if there are any other objects, it receives another index. This is repeated until all objects are checked.

Master sends to all slaves indices of objects that have to be added to cluster, and removed from $typical$.

Searching is continued as long as there were any object added to cluster.

If no more objects were added, each slave clears its cluster. All the processes compute typicality of their part of objects that remain in $typical$. Master receives it and accumulates, creates next cluster containing most typical object and so on. This is continued until there are any objects in $typical$ and most typical object has a typicality not less than β.

In this algorithm similarity between each two objects is calculated many times. We had an idea of creating similarity matrix of objects. Such a solution is good with small populations. In the case when the population size is large, similarity matrix is too big to fit in memory. For example: if size of the population is 250 000 then similarity matrix takes the place of $250\ 000^2 * \text{sizeof}(float) = 250\ 000^2 * 4$ bytes. It is over 232 GB. At this moment this is too large to fit into memory. On the other hand keeping it on hard disk is not profitable because of disk access time.

Algorithm 2 TypicalClustersMaster(P, p, Q, Sim, A, B)

```
Require: $P-$ population, $p-$ number of processes ($p \geq 2$), $Q-$ pointer to function of fuzzy quantifier,
    $Sim(o_i, o_j)$ - similarity function $o_i, o_j \in P$, $A$, $B$ - functions counting value of factors $\alpha$ and $\beta$
 1: pack population $P$ and send it to all slaves
 2: if $P.size$ mod $(p-1) > 0$ then
 3:     for $i = N - (N \bmod (p-1))$ to $N-1$ do
 4:         $o_i.t = typicality(o_i, P)$   $typical.Add(o_i)$
 5:     end for
 6: end if
 7: receive from slaves and save in $typical$ typicality of objects they calculated
 8: send typicality of all objects to all slaves
 9: find $o'$ the most typical object {the object with the highest degree of typicality}
10: $\alpha = A(o'.t)$   $\beta = B(o'.t)$   $k = 0$
11: while $typical.Size > 0$ and $o'.t \geq \beta$ do
12:     create new cluster $C_k$
13:     $C_k.Add(o')$   $typical.Remove(o')$
14:     repeat
15:         send to each slave a consecutive index of object {Slave counts similarity of this object to the
            cluster $C_k$}
16:         while any slave count similarity do
17:             receive from any slave $s_x$ similarity of object $o_i$ and if it is similar save $i$ in the array $Inds$
18:             if there are indices, that wasn't send then
19:                 send consecutive index $i$ to slave $s_x$.
20:             end if
21:         end while
22:         send to all slaves array $Inds$.
23:         for all objects $o_i : i \in Inds$ do
24:             $C_k.Add(o_i)$   $typical.Remove(o_i)$
25:         end for
26:     until $Tab.Size > 0${there where any objects added to cluster $C_k$}
27:     if $typical.Size < p$ then
28:         send message with tag=11 to redundant slaves {Slave exits.}
29:         $p = p - typical.Size$ send $p$ to other slaves
30:     end if
31:     $N = typical.size$
32:     if $N$ mod $(p-1) > 0$ then
33:         for $i = N - (N \bmod (p-1))$ to N-1 do
34:             $o_i.t = typicality(o_i, typical)$
35:         end for
36:     end if
37:     receive from slaves and save in $typical$ typicality of objects they calculated
38:     send typicality of all objects to all slaves
39:     find $o'$ the most typical object
40:     $\alpha = A(o'.t)$   $\beta = B(o'.t)$   $k = k+1$;
41:     send $\alpha$ and index of $o'$ to all slaves
42: end while
43: return clusters $C_0 \dots C_n$
```

Algorithm 3 TypicalClustersSlave$(p, p_i d, Q, Sim)$

```
Require: $p-$ number of processes ($p \geq 2$), $p_{id}$ - rank of this process, $Q-$ pointer to function of quantity
    in agreement,$Sim(o_i, o_j)$ - similarity function $o_i, o_j \in O$
 1: receive and unpack population $original$
 2: $N = original.Size$
 3: for all $o_i$, such that $\frac{N}{p-1} * (p_{id} - 1) \leq i \leq \frac{N}{(p-1)} * p_{id}$ do
 4:     $T[i] = typicality(o_i, original)$
 5: end for
 6: send filled part of $T$ to master
 7: receive from master index $m$ of the most typical object $o' = o_m$
 8: receive from master array $T$ {typicality of all objects}
 9: create new population $typical$ from objects of $original$ and typicality from array $T$
10: create new cluster $C$ with most typical object $o'$
11: remove $o'$ from $typical$
12: repeat
13:     test for a message from master
14:     if message.tag = 11 then
15:         exit;
16:     else if message.tag = 4 then
17:         receive $\alpha$ from Master
18:     else if message.tag = 6 then
19:         receive index $i$ from master
20:         check if $o_i$ is similar to the cluster $C$ and send similarity to master
21:     else if message.tag = 7 then
22:         receive array $Inds$ from master
23:         if $Inds.size > 0$ then
24:             for all objects $o_i : i \in Inds$ do
25:                 $C.Add(o_i)$;    $typical.Remove(o_i)$;
26:             end for
27:         else {there are no objects similar to cluster.}
28:             if $typical.size < p - 1$ then
29:                 receive $p$ from master or exit if message.tag = 11
30:             end if
31:         end if
32:         N=typical.Size
33:         for all $o_i \in typical$, such that $\frac{N}{p-1} * (p_{id} - 1) \leq i \leq \frac{N}{(p-1)} * p_{id}$ do
34:             $T[i] = typicality(o_i, typical)$
35:         end for
36:         send filled part of $T$ to master
37:         receive from master index $m$ of the most typical object $o' = o_m$
38:         receive from master array $T$ {typicality of all objects in $typical$}
39:         for all $o_i \in typical$ do
40:             $o_i.t = T[i]$
41:         end for
42:         $C.clear$    $C.Add(o')$    $typical.Remove(o')$
43:     end if
44: until message.tag $\neq$ 11
```

6. Results of experiments

We tested our algorithm on Compute Cluster - ”mordor2” at Faculty of Computer Science of Bialystok Technical University. There are 16 compute nodes, each of 2 quad-core processors Intel Xeon X5355 (2660 MHz) with 4GB RAM. The program is implemented in C++, and we use Intel compiler.

We performed three experiments with three various population size. In the first experiment, there was 65 536 objects (44 192 unique values of attribute on witch similarity function was defined) and 190 clusters were created. In the second 301 clusters from population of 262 144 objects (69 997 unique values). The last experiment was performed on population 1 064 000 objects. This time only 28 clusters were created , because there was only 254 unique values of attribute. We used factors: $\alpha_p = 75\%$ of the most typical value ($\alpha = o'.t * 75\%$) and $\beta = 0$ to cluster all the objects. Figures 1, 2 ,3 show speedup of our parallel algorithm for those three experiments and Table 1 contains time in seconds of performing each of this three experiments.

In all figures p is the processors number, $T_1(p)$ - is time of processing first experiment on p processors, $T_2(p)$ and $T_3(p)$ second and third experiment accordingly. $T(1)$ - is the time of running program sequentially on one processor. Speedup of algorithm is calculated by formula in eq. 5.

$$S(p) = \frac{T(1)}{T(p)} \tag{5}$$

Analysing the charts we can say that there is no speedup for 2 processors because of algorithm architecture. One of two processes is master and it doesn’t take part in calculations. From 5 processors to 20 processors (fig. 2) we can say that speedup is close to linear speedup. It grows slower for greater number of processes as a consequence of Amdahl’s law. It starts to saturate earlier (with less number of processors) for smaller population (fig. 1 and later for greater population (fig. 3). Like in fig. 3 the chart locally drops. It may be caused by load imbalance - there are situations when master participates in calculations or not, depending on the number of processors and actual number of objects. It is noteless with smaller population (fig. 1). The time of running program on one processor in third experiment is 73 638s. that is over 20 hours, while on 48 processors 1 760.04s - somewhat less than half an hour.

7. Conclusion

The parallel version of this algorithm allows to deal with large data bases in acceptable time. The more objects we have, the more processors we can use without

Table 1. Time of performing experiments

p	$T_1(p)$	$T_2(p)$	$T_3(p)$	p	$T_1(p)$	$T_2(p)$	$T_3(p)$
1	536	8833	104 066	25	28.024	392.796	2862.57
2	542.771	8827.04	111943	26	26.6639	398.204	3086.98
3	440.099	6917.32	76519.8	27	26.1264	384.812	2973.57
4	369.37	5938.48	41201	28	25.2003	375.33	2885.53
5	294.841	2232.53	16613.9	29	24.4827	341.483	2482.13
6	160.135	1786.62	13615	30	23.8073	331.278	2425.07
7	93.7401	1495.43	11613.9	31	23.3445	341.783	2724.54
8	81.7998	1372.49	11135.7	32	22.8769	330.902	2649.3
9	72.1456	1104.92	24866.1	33	22.1609	321.069	2430.05
10	63.8581	991.029	14712.6	34	21.681	310.859	2358.91
11	57.6498	899.491	6788.27	35	21.3022	304.901	2303.22
12	52.7288	820.887	6253.87	36	20.7538	279.391	3606.57
13	48.8878	761.468	5829.14	37	20.7321	305.128	2282.51
14	45.498	746.249	5958.49	38	20.3226	299.343	2219.04
15	43.1952	691.14	9235.31	39	20.0423	293.134	2162.86
16	40.291	684.79	5434.6	40	19.6307	284.499	2113.97
17	38.0155	572.078	4291.65	41	19.1846	263.208	1953.81
18	36.14	541.105	4086.26	42	18.8672	257.352	1910.82
19	34.937	510.345	3774.39	43	18.8057	242.463	1965.2
20	33.3961	484.716	6627.98	44	18.4836	260.475	1921.94
21	31.7411	493.387	3862.84	45	18.3377	232.151	1879.08
22	30.6009	495.573	3874.26	46	18.1134	250.914	1834.79
23	29.4827	476.159	3699.19	47	17.8429	246.288	1794.08
24	28.4549	456.875	3556.91	48	17.6971	242.1	1760.04

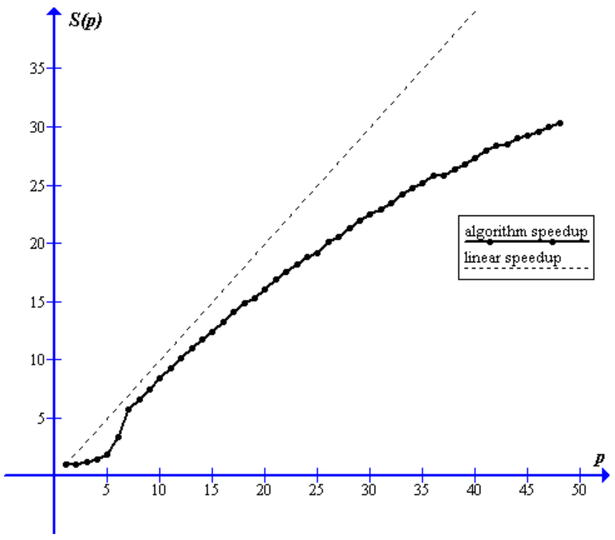

Fig. 1. Algorithm speedup for population of 65 536 objects

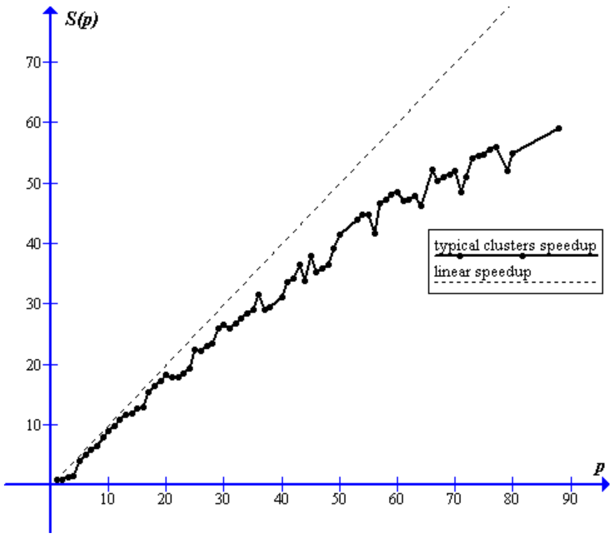

Fig. 2. Algorithm speedup for population of 262 144 objects

efficiency degradation. Our algorithm is not perfect. We use only blocking communication operations and there is moment of poor load balancing (master should more participate in computation). There are also messages that master sends to all pro-

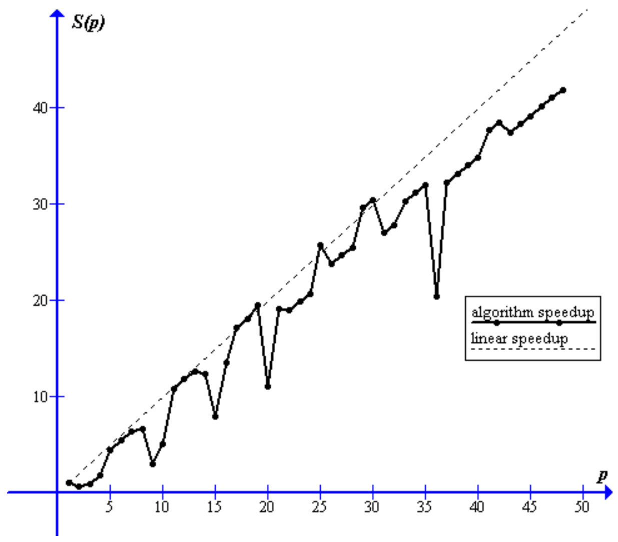

Fig. 3. Algorithm speedup for population of 1 064 000 objects

cesses. It would be better to use collective communications operations. We have not implemented those yet.

We didn't make parallel linguistic summaries, but only clustering algorithm, because complexity of the former is $O(N)$. Object-oriented databases, that are subject of our interest, allow creating object attributes that are collections of other objects or other data like multimedia. In that situation, predicate defined on such an attribute or group of attributes may be very time consuming. Next we plan to implement parallel linguistic summaries to deal with such databases.

References

[1] Smolińska M. Sosnowski Z.: Linguistic Summaries with Fuzzy Clustering, Zeszyty Naukowe Politechniki Białostockiej. Informatyka Nr 2 (2007), s.141-154 (in Polish)

[2] Rasmussen D.: Application of the Fuzzy Query Language - Summary SQL, DATALOGISKE SKRIFTER, Roskilde University, 1997.

[3] Rasmussen D., Yager R.: Introduction of Fuzzy Characteristic Rules by Typical Values, DATALOGISKE SKRIFTER, Roskilde University, 1997.

[4] Grama A., Gupta A., Karypis G., Kumar V.: Introduction to Parallel Computing, Second Edition, Addison Wesley, 2003.

Marta K. Smolińska, Zenon A. Sosnowski

PODSUMOWANIA LINGWISTYCZNE Z RÓWNOLEGŁYM GRUPOWANIEM ROZMYTYM

Streszczenie: Z podsumowaniem lingwistycznym, jak i z predykatem rozmytym związana jest wartość prawdy. Możemy więc podsumowań lingwistycznych używać jako predykatów rozmytych. Podsumowanie postaci "większość obiektów w populacji P jest podobna do obiektu o_i wykorzystać możemy do znajdowania typowych wartości w populacji P, które to wykorzystuje rozmyty algorytm grupujący. Wadą tego algorytmu jest jego duża złożoność obliczeniowa. W celu przetwarzania dużej liczby danych zaimplementowaliśmy ten algorytm równolegle, korzystając ze standardu MPI do komunikacji między procesami działającymi na różnych procesorach. W tej pracy przedstawiamy algorytm równoległy i wyniki eksperymentów.

Słowa kluczowe: podsumowania lingwistyczne, grupowanie rozmyte, programowanie równoległe

Artykuł zrealizowano w ramach pracy badawczej S/WI/2/08.