8-2008

Service oriented transitive closure solution

Jonathan Baran
University of Arkansas, Fayetteville

SERVICE ORIENTED TRANSITIVE CLOSURE SOLUTION

By

Jonathan Baran

August 2008
University of Arkansas

Table of Contents

Abstract

The goal of this project is a service based solution that utilizes parallel and distributed processing algorithms to solve the transitive closure problem for a large dataset. A dataset may be view conceptually as a table in a database, with a physical structure representing a file containing a sequence of records and fields. Two records are said to be transitively related if and only if they are directly related due to sharing of one or more specific fields, or a sequence may be made from one record to the other under the condition that all intermediate entries are related the immediate previous and subsequent entry. The transitive closure problem is to cluster the records in a dataset into groups such that all transitively related records are in one group.

An approach to solve this problem is to divide the task into two separate problems. The first of these problems is the processing of the dataset, and thus generating a set of pairs. Each of these pairs would include two record identifiers, and these pairs would exist if and only if these two records were directly related. The second of these problems is to use the record pairs to cluster the records into transitive closures. The current software solution solves this second sub problem through the reading of record pairs, produced by a different software solution, and writes the completed results of the transitive closure problem to a file.

This thesis studies how to enhance the current software solution in such a way that it becomes a “service”. The study includes designing, implementing, testing, and evaluating the enhanced solution. The service model identifies an aspect that would potentially benefit from restructuring or addition of functionality. A current issue is the lack of an ability to fetch transitive closure from within the solution upon the completion of a job, and is thus limited in its direct use with other processes or applications.

1. Introduction

Data processing and storage is an important part of many businesses. The goals of handling of data are to keep the information accessible, accurate, and to serve the information in a reasonable form. Steps that may assist in aiding any of these aspects of data management can be beneficial to any process or business that will need to deal with this information in the future.

In the retrieval of data, such as for data quality assurance [1] or data mining [2], it is beneficial to be able to analyze and organize the data based upon relationships between the data present. For the purpose of this research, the data contents of a dataset will be called entries, and these entries will have properties that called relationships, designating that two entries share logical commonality, such as sharing key entry fields, that is useful for the purpose of organization. An entry may either have or not have a relationship with another entry.

More than one relationship is permitted for any given entry, allowing a single entry, designated entry 'A', to connect two or more separate entries, designated entries 'B' and 'C'. In this condition, should the 'B' and 'C' not share a relationship with each other, they may be referred to as transitively related to each other through entry 'A'. Such transitive relationships may extend for additional entries. Should entry 'C' have additional relationships beyond those with entry 'A', these will also be transitively related to the afore mentioned entries. This technique of identifying data commonalities using transitive relationship of fields is just one of the methods used for identifying groups of entries that describe a common theme or logical entity. For other methods, the reader is referred to [3].

The combined total of all transitively related entries for any given entry is called a transitive closure. Once given the definition of a relationship, any dataset may be broken into

distinct transitive closures based upon this rule. This project investigates a method for performing this process over a dataset that has been extended over multiple machines into a grid or cluster format [4] followed by using a service interface to this transitive closure solution [5] that allows for the retrieval of a distinct transitive closure.

2. Background

This research adds a service feature to a project researched by Donald Hayes [9], using a grid provided by Acxiom, written using the C++ CORBA library. The grid consists of multiple dual-core, 1.2 gigahertz processors, each with 4 gigabytes of RAM and running the CentOS Linux operating system.

ID	Parent ID	First Name	Last Name	State	Address
0	2	Jim	Jones	TX	555 Solid St
1	9	James	James	TX	575 Green St
2	1	John	Jones	TX	555 Solid St
3	0	John	Smith	TX	575 Green St
4	6	Jane	Jones	TN	565 Oak Rd
5	7	Sarah	Smith	TN	454 Red Rd
6	11	Jane	James	TN	645 Castle St
7	8	Jane	Jones	TN	565 Oak Rd
8	4	Laura	Smith	TN	645 Castle St
9	10	Jim	Jones	TX	575 Green St

Fig 1. *Example data representing two families*

The task of calculating transitive closures may seem daunting, but it can be broken into several steps. The first of these is to take an initial dataset (Fig 1) and create direct-pair links. Given a large set of initial data, the area of interest may be refined a specific area, and within this specific area it may be further confined to only specific data of interest. For the purpose of illustration, Figure 1 is presented, containing data regarding two family histories.

Initial processing converts this refined data into a minimum form of data that is required for the processing phases. Regardless of the size of an entry, or the number of key fields for comparison that it possesses, its relationship with other entries may be represented by only noting that this pairing between the entries exists. Figure 1 presents extraneous data that is not

needed for calculating family data, as all that is needed is the parent- child relationship data is the first two columns specifying who is a parent and who is a child.

ID	Name
0	Son
1	Grandfather
2	Father
3	Grandson
4	Grandmother
5	Granddaughter
6	GreatGrandmother
7	Daughter
8	Mother
9	GreatGrandfather

Fig 2. *A mapping of alpha-numeric keys to table entries in Figure 1*

Left	Right
0	2
0	3
6	4
9	1
8	3
8	7
1	2
1	9
3	0
5	7
4	8
4	6
7	5
7	8
2	0
2	1

Fig 3. *Pairs generated based on the parent-child relationships in Figure 1, using the mappings in Figure 2*

To reduce and abstract this concept to a greater extent, you may represent each entry in a dataset by an alpha-numeric value (Figure 2), such as an ID or key in a database, and may signify a relationship between two entries as a pair of these two ID (Figure 3). For the purpose of human readability, Figure 2 proposes a mapping of ID for each entry. Figure 3 illustrates that parent-child pairing present in Figure one, such as entry 0 being related to both entry 2 as a child, and entry 3 as a parent.

This method restricts the data that must be sent over networks, processed by programs, or stored in memory [6, 8]. The process of converting a pair of related entries to a pair IDs may be reversed at any time. It is assumed for this report that there exists a method to generate these pairs in parallel, and that when processing of transitive closures begins a data source has been converted to directly related pairs. These pairs are then read into the system, and each entry is given a numeric key [6] that it may be symbolically referred throughout the formation process. For all purposes 'numeric key' and 'entry' may be used interchangeably from this point on.

From this point the individual pairs must be processed to form transitive closures. One may think that this would be an N^2 operation to group all ID pairs, involving checking of every pair with every other pair in the dataset. However, this issue may be solved using the Union-Find algorithm [7], based upon creating of a representative group key, looping through the collection of entries, and tying all related entries to a shared group. After one iteration, all items within the collection will be tied to a transitive closure, with an expense of only $O(\alpha(n) \cdot n)$ time, where $\alpha(n)$ is the inverse of Ackermann the function, or at most 4 for all reasonable values of n.

This solution works for a database that may be loaded into a single computer's memory, but there are situations where the memory or processor need for this calculation can not be handled by a single machine. In these cases, multiple computers may be used to allow for expanded memory capacity. This provides the ability to implement parallel processing, using network based communication on a cluster.

The involvement of a cluster in this project requires a new technique be implemented to solve the disjoint transitive closure solution, taking into account that each machine in the cluster is not able to freely access all memory involved in the solution [5].

Process 1	
Local	Connected
0	2
0	3
6	4
9	1
8	3
8	7

Fig 4a

Process 2	
Local	Connected
1	2
1	9
3	0
5	7

Fig 4b

Process 3	
Local	Connected
4	8
4	6
7	5
7	8
2	0
2	1

Fig 4c

Fig. 4 *The distribution of pairs across all three processes in the cluster*

Each machine in the cluster will be assigned a set of the record entries in the dataset, so that the ownership of the entries is balanced among all participating machines (Figure 4a-4c). For example, Process 1 has entries 0,6,8, and 9 locally. Each entry can be imagined as nodes in a graph, each possessing potential connections to other nodes. The decision for a machine to have an entry locally is performed using some method of hashing to aid in equal distribution of the entries among all machines.

Process 1			
Left	Right	Group	Local/Global
0	2	0	Global
0	3	0	Global
6	4	6	Local
9	1	9	Global
8	4	6	Local
8	7	6	Global

Fig 5a

Process 2			
Left	Right	Group	Local/Global
1	2	1	Global
1	9	1	Global
3	0	3	Global
5	7	5	Global

Fig 5b

Process 3			
Left	Right	Group	Local/Global
4	8	4	Global
4	6	4	Global
7	5	7	Global
7	8	4	Global
2	0	2	Global
2	1	2	Global

Fig 5c

Fig. 5 *The data contents of each process after the first iteration of grouping*

The processed data pairs that were created at the start of this process by an external application will then be distributed to all of the machines. Should a machine own a set of entries locally, it will receive all pair data for which at least one end is a local entry. This pair data may associate the entries on this machine with other local entries on the machine, which can be thought of local edges between the machine's nodes. The nodes may also have relationship with nodes that exist on other cluster machines. These can be thought of as global edges in the system, and nodes that are connected by edges will be considered to be in the same group (Figure 5a-c).

In this process [5], each of the nodes will apply the Union-Find algorithm locally to process all local edges. It will then use this information to identify which elements within this local transitive closure have relationships with other machines. It can then inform that if two global edges with a group locally, then they should inform the corresponding machines of this relationship. In the figures above, an example of this formation is that Process 1 has entry 2 and entry 3, are within group 0, so it may send this pair information to the other processes. This process is done by all machines, using information they calculate locally to create new edges for other for continual grouping.

Process 1	
Left	Group
0	0
6	6
9	0
8	6

Global Edges			
Left	Group	Remote Key	Processor
0	0	1	2
0	0	2	3
6	6	5	2
6	6	4	3

Fig 6a

Process 2	
Left	Group
1	1
3	1
5	5

Global Edges			
Left	Group	Remote Key	Processor
1	1	2	3
1	1	0	1
5	5	4	3
5	5	6	1

Fig 6b

Process 3	
Left	Group
4	7
7	7
2	2

Global Edges			
Left	Group	Remote Key	Processor
4	4	6	1
4	4	5	2
2	2	1	1
2	2	0	2

Fig 6c

Fig. 6 *The data contents of each process after all transitive closures have been formed.*

This process is used to form new local relationships on machines, which in turn create new global edges. After several iterations of this process, all machines within the system have located all local nodes in a system that are within a transitive closure, and have formed all global edges that connect these nodes to other machines (Figure 6a-c). Through the formation and union of groups on the clusters, two distinct groups are figured that are spread over all three of the processes.

3. Problem

Once the data has been processed by the machine in the process, it will output information regarding all of the transitive closures that were found. The illustrated example would contain three separate outputs, providing a text based output. This is a single bulk output that provides all of the contents of the edges formed in the memory of each individual machine in the cluster, and provides no way to filter the data into a usable form for any external service. The solution at this point is lacking in any communication protocol with an external application, and does not have an internal system set up that can fetch and process requests regarding transitive closures should a request reach the application.

For small datasets, one may be able to analyze the results visually or though use of a search application, but when a dataset has grown to a significant size, such as processing of millions of entries, this solution becomes increasingly unfavorable. Not only does this solution provide excess information that is not needed for many queries, but it requires use of another application, hard drive space, and additional time to read this file and process it for consumption by another service.

In a business setting, an ideal service would need a method that would allow a user or service to provide some key or other identifying trait of a dataset entry, which would then be processed and provide all other entries related to the closure. Such features require an enhancement of not only input and output functionality to each of the processes in the cluster, but an internal structure change that allows each of the clusters to exam and handle the internal storage in a way that understands the request and provides complete and accurate results.

4. Approach

An ideal solution would be a method to access only the required data, and provide only the results of a single closure. Currently each process in the application has the current number of processes in total, a set of methods to call remote procedures within other processes, and the hashing function used to distribute provided alpha-numeric IDs among all available processes. This is vital to the service, due to the fact that should a transitive closure exist with the provided entry as a member, it would be located on this process.

With regards to transitive closures, each process retains all the previous solution's data upon outputting of the closure information, including all of the local entries grouped into transitive closures, and global edges for each of these closures that provide both the process that is responsible for the remainder of a closure, and an ID that aids in locating this remote group (Fig 6a-c). These entries are stored in their numeric key form, and each process also retains its ability to convert the data back into its alpha-numeric form.

To begin a fetch of a transitive closure from this system, an entry is needed. This data must match the alpha-numeric value used by the initial pair providing service, so that it may hash this information and locate the process responsible for the entry should it exist in the dataset. Through its remote procedure methods, any queried process in the system may contact the responsible process and forward the request to this machine. An example using the previously provided data is a fetch for "Grandfather", entry 1 in Figure 2.

At this point in the solution, should the queried entry exist in the dataset, the process responsible for this entry has received an initial request for a transitive closure. Process 2 has received this request, and we can observe that "Grandfather" is in the dataset mapped to entry 1

(Figure 5). Using the local entry information, the process may identify the group that contains the entry and locate any global edges that exist for this group. Combining this information, it will then return a list of processor ids and numeric keys for each group component of the transitive closure: Processor 1 holding entry 0, and Processor 3 holding entry 2.

This data is the minimalistic information needed to fetch the complete transitive closure. Given a large dataset, with potentially numerable processors, this function will return only the processes and IDs required to fetch a complete closure.

At this point, parallel processing may be implemented in the fetching segment. Given a process number and a numeric key to a closure segment on said processor, no intercommunication is needed between the processes for the remainder of the fetch. The querying can contact the provided processes using their process id, and send the numeric key corresponding to the remote closure. The process will received this request, and will then locate the corresponding closure's group within its local data store. The processes will search through their local data to find all other entries that belong to this closure and reverse the numeric key to the original alpha-numeric value, with Process 1 returning entry 0, “Son”, and entry 9, “GreatGrandfather”; Process 2 returning entry 1 and 3, “Grandfather” and “Father”; and Process 3 returning entry 2, “Grandson”. The data will then be returned to the querying application. The query application may then collect and process this data.

5. Testing and Results

For the purpose of testing, a dataset was selected in the range of 2 million entries, and 7.5 million edges. This data was processed by the transitive closure solution, and subsequently queried by the service addition to select a known closure of 1,472 members for the purpose of timing comparison.

The timing method was composed of repeating a request for an entry 1,000 times and recording the average request timer for the fetch process. This was performed for multiple members of the closure, and using one to six processors in the grid.

The process was timed for both the initial process fetch for the processor id and remote key pairs, and for the subsequent queries of transitive closure components. The initial fetch was noted as negligible, as it did not require more than one value search into its local data, and returned at most 2n integers, with n being the number of processes in the system.

The time spent for a query to each processor was averaged for the 1000 attempts, and then grouped by processor count to record the average amount of time that was spent on each processor.

Processor Count	Average Time Per Process	Average Total Fetch Time	Speed Gain ($Time_n$ - $Time_{n-1}$)
1	.0420 seconds	.0420 seconds	------
2	.0365 seconds	.0730 seconds	.0055 seconds
3	.0240 seconds	.0720 seconds	.0125 seconds
4	.0170 seconds	.0680 seconds	.0070 seconds
5	.0112 seconds	.0560 seconds	.0058 seconds
6	.0101 seconds	.0608 seconds	.0011 seconds

Fig. 7 *Testing results based on queries to a 2 million entry dataset (1472 entries retrieved)*

The decrease in time per-fetch of closure fragments appears to diminish with each additional processor added to the solution. This was performed with the expectation that there was not a linear reduction, given an understood overhead added due to create and process time for each additional connection.

6. Future Work

The current setup requires a use of a hard drive, creating a bottleneck in execution. A network disk-drive is created that all processes may access for the duration of the clustering and data fetch events. Each process in the solution currently relies on sharing of remote interfaces through writing and reading of serialized interface files to this drive. Each process will read and de-serialize this file, and then will have the ability to contact and call remote functions of other processes.

The hard drive also plays a role in the storage of the initially provided entry pairs. It must be read by all processes, during which time it will used for the hashing, numeric conversion, and distribution of the data. If there existed a process to send the required data by use of network protocol or other means, this would eliminate the need for a large network share, or for the extensive time currently used to fetch and load potentially sizable amounts of data.

With regards to usability the application, the service constructed currently possesses the ability to provided queried information, but only offers a minimal interface for retrieval. Future works may apply or improve this internal system, expanding or enhancing the input or output interfaces as required. One such potential improvement would be the creation of an on-cluster service that acts as a mediator to outside processes. This may act as a layer for various protocols, or may itself be a process that would allow for the aforementioned closure parallel retrieval of data, thus reducing the need for sequential retrieval.

7. Conclusion

Through the experiments, the transitive closure solution was found to be usable for fetching of complete closure sets. Based upon sequential processing of the processor ID and remote key data during testing, the results do not show a noticeable change in total retrieval time with the addition of processes to the solution. This appears to only add additional overhead to the retrieving of the set count of entries. The decrease in time per-fetch of closure fragments appears to diminish with each additional processor added to the solution. This was performed with the expectation that there was not a linear reduction, given an understood overhead added due to create and process time for each additional connection. These results present a potential speed increase in fetching of larger datasets though parallel retrieval.

References

[1] M. Hernandez, S. Stolfo, Real-world Data is Dirty: Data Cleansing and the Merge/Purge Problem, Journal of Data Mining and Knowledge Discovery 1(2), 1998.

[2] T. Redman, Data Quality for the Information Age, Artech House, Norwood, MA., 1996.

[3] R. Baxter, P. Christe, T. Churches, "A Comparison of Fast Blocking Methods for Record Linkage" ACM SIGKDD '03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation, Washington, DC, 2003, pp 25-27.

[4] W. Li, T. Schweiger, "Distributed Data Structures and Algorithms for Disjoint Sets in Computing Connected Components of Huge Network" Proc. International Conference on Parallel and Distributed Processing Techniques and Applications, Volume II, 905-909, 2007.

[5] W. Li, D. Hayes, J. Zhang, R. Bheemavaram, C. Portor, and T. Schweiger, "Parallel and Distributed Grouping Algorithms for Finding Related Records of Huge Data Sets on Cluster Grids," Proc. Acxiom Laboratory for Applied Research (ALAR) 2005 Conference on Applied Research in Information Technology, March 2007.

[6] R. Bheemavaram, J. Zhang, and W. Li "A Parallel and Distributed Approach for Finding Transitive Closures of Data Records: A Proposal," Proc. Acxiom Laboratory for Applied Research (ALAR) 2005 conference on Applied Research in Information Technology, March. 2006, pp. 61-70.

[7] T. Cormen, S. Clifford, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, MIT Press, 2001.

[8] R. Bheemavaram, "Parallel and Distributed Grouping Algorithms for Finding Related Records of Huge Data Sets on Cluster Grids", MS thesis, University of Arkansas, 2006

[9] D. Hayes, "Transitive Closure of Data Records: Application and Computation", MS Thesis, University of Arkansas, thesis pending