A File-Based Approach for Recommender Systems in High-Performance Computing Environments

Simon Dooms, Toon De Pessemier, Luc Martens
WiCa group, Dept. of Information Technology, Ghent University
Gaston Crommenlaan 8 box 201, B-9050 Ghent, Belgium
Email: {Simon.Dooms, Toon.DePessemier, Luc.Martens}@intec.ugent.be

***Abstract*—Since recommendation systems tackle the problem of information overload, the processing of huge datasets can not be avoided. When these datasets no longer fit into the RAM memory of a computing node, a scalable data storage approach is required. While database systems are frequently used for this goal, they have their disadvantages and when not properly designed may slow down the recommendation process. In this paper we propose an alternative file-based data storage approach that is particularly well suited for a high-performance computing environment where the usage of databases may not always be an option.**

By breaking down the recommendation process in separate phases and carefully structuring the input and output of each phase, we have build a file-based recommendation system that scales proportional with the number of computing nodes and processor cores available in each node.

***Keywords*-recommender system, file-based, high-performance computing, HPC, scalable, NoSQL**

I. Introduction

Recommender systems try to counter information overload by sifting data and extracting only bits and pieces relevant to the user. This implies recommendation algorithms have to process very large datasets. Working with often millions of users and items rules out the strategy of simply reading all data into RAM memory and start processing. Efficient data storage is therefore required to provide fast data access with minimal delay.

Classical relational database management systems (RDBMSes) are often put to the task [1][2] although they may not always be the best option. The required data throughput needed by recommendation algorithms is very high. Massive amounts of small intermediate values must be stored and retrieved during execution time. If for every required value a data connection with the database needs to be set up and closed down, the accumulated resulting data delays would make out most of the total execution time altogether. More optimized approaches may be to fetch large chunks of data at once to minimize database interaction. The most optimal being probably to prefetch as much data as can possibly fit in RAM memory. This leads to an interesting idea. If it is so important to keep interaction with the database low, then why not try to leave out the database entirely? It would free developers of the sometimes cumbersome tasks of designing efficient database structures, creating indexes and maintaining the database management software.

Many alternatives to the classical database, often referred to as NoSQL systems, have been developed and some even found their way into the recommender systems domain [3]. We want however an alternative that easily maps on the HPC infrastructure we have at our disposal (see section II) or for that any interconnected network of nodes with both local and shared storage capacity.

We believe scalability and the concept of keeping the data close to the work are the main goals for data storage optimized for recommender systems. File systems like the Hadoop Distributed File System (HDFS [4]) look promising but they often require a complete reorganization of the recommendation algorithm. Hadoop for instance offers a fast and fault-tolerant distributed computing environment but demands that the algorithm is implemented in terms of *map* and *reduce* operations, rendering it useless in any other environment [5][6]. Furthermore, we did not want to be bound by what technologies the infrastructure supports and so we looked into the most obvious data storage approach of all.

The most straightforward way to store data on a system is by means of files. We found the file-based approach satisfying our needs for easy file system migrations, scalability and performance by structuring input and output files as described in the following sections.

II. High-performance computing infrastructure

Fig. 1 shows the conceptual layout of one of the clusters (called *gengar*) of the high-performance computing infrastructure we have at our disposal. Gengar has 194 computing nodes each of which contains 8 cores at 2.5 GHz, 16 GB RAM and 146 GB of local storage capacity. Every node is also networked to shared storage in a RAID5 configuration.

III. The Recommendation workflow

In contrast to much literature about recommender systems this paper will not focus on the internals of the recommendation algorithm. Instead the main focus will be on how we can provide the algorithm with the required data and

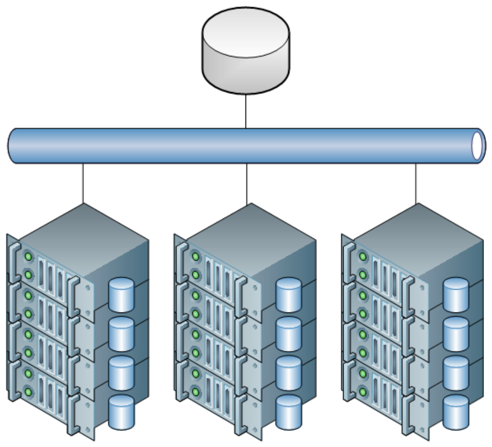


Figure 1. The conceptual layout of the high-performance computing infrastructure at our disposal. Every computing node disposes of a local hard disk and has access to a shared storage device in the network.

structure files such that minimal read (and write) delays can be traded off with scalability. To show our approach works with any type of recommendation algorithm we employ a hybrid of content-based (CB) and collaborative filtering (CF) much like described by Cornelis et al. [7]. Since this hybrid algorithm internally mixes CB and CF, the processing of the required input for both types of algorithms can be demonstrated. As stated, we will however make an abstraction of the algorithm itself and provide no further in-depth information.

We abstract the recommendation process to a three-phase workflow that requires the inputs and outputs as shown in Fig. 2. The process starts off with the availability of item metadata and consumptions of users. User consumptions can be anything from explicit feedback by means of star-ratings to implicit feedback like behavioral logfiles.

In a first phase, item similarity is calculated. The item similarity can rely on item metadata, user consumptions or both. The resulting item similarities in their turn serve as the input of the user similarity calculation together with the user consumptions. In the final phase all three user consumptions, item similarities and user similarities are used to generate the recommendations. For a non-hybrid recommendation algorithm this three-phase workflow can be reduced to two phases by leaving out the first (if CF) or second phase (if CB) according to the input requirements for the specific algorithm.

The following sections will show how the input and output of each phase can be structured to allow a file-based and scalable data handling approach.

A. Phase 1: Item Similarity

Item similarity has been a hot topic in the information retrieval domain for several years. The problem to be solved is how similar two given items in a dataset are. There

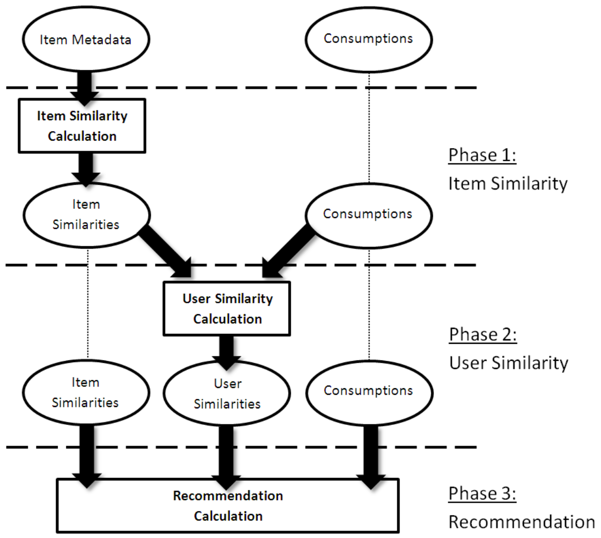


Figure 2. The abstracted workflow of the recommendation process focused on the ins and outs of every phase.

are numerous ways to go about this, the most popular ones are Cosine Similarity and Pearson Correlation [8] but many more have been researched. Again we will make an abstraction of the problem, and assume we have an algorithm that given two items calculates their similarity. We focus on the problem of matching every item in the dataset and providing the necessary input to the algorithm.

Table I shows the item similarities for 5 items. Every item must be compared to every other item, but item similarity is symmetric so only the half size triangular matrix needs to be calculated. Since an item is equal to itself, the total number of comparisons that need to be done will be $\frac{n_i(n_i-1)}{2}$ with n_i the number of items.

Table I
THE ITEM SIMILARITIES FOR 5 ITEMS

	i_1	i_2	i_3	i_4	i_5
i_1	x	$0_?$	$1_?$	$2_?$	$3_?$
i_2	x	x	$4_?$	$5_?$	$6_?$
i_3	x	x	x	$7_?$	$8_?$
i_4	x	x	x	x	$9_?$
i_5	x	x	x	x	x

The non-scalable approach would be to read all the metadata from one big file into memory and start comparing items. More scalable is to divide the similarity comparisons amongst the available nodes (and cores). To do this, we project the calculation jobs in a one-dimensional space as follows.

$0_?$	$1_?$	$2_?$	$3_?$	$4_?$	$5_?$	$6_?$	$7_?$	$8_?$	$9_?$

Parallelizing the item similarity calculation is now a matter of splitting the one-dimensional job list in m equally large chunks (with m the number of nodes available). For every node, jobs can be split up even further between the available cores per node. Below an example of the parallelizing of item similarity with 5 items over 5 nodes with each 2 cores.

$0_?$	$1_?$	$2_?$	$3_?$	$4_?$	$5_?$	$6_?$	$7_?$	$8_?$	$9_?$

Splitting up the calculations in this way, turns the problem of item similarity into an embarrassingly parallel problem with very few dependencies between the jobs. In fact the only thing that the jobs share is the input data. Job $0_?$ will need the metadata of items i_1 and i_2, job $1_?$ from items i_1 and i_3 and so on. If sufficient RAM is available in a node, all the metadata can be loaded. Otherwise a slicing of the metadata will be required to make sure every node is capable of loading its data. For our dataset of 53,000 items (see section IV) the metadata was but 50 MB and could easily be loaded.

The (file-based) output of the item similarity phase should be carefully structured so that easy and efficient accessibility is possible in the next phase. Since the output growth of item similarity is quadratic by nature, disk usage will rapidly increase. For the similarity of 53,000 items over a billion comparisons must be calculated and stored.

Two extreme options would be dumping all calculated similarities in one big file or writing to a new file for every item, none of which are scalable. In the latter case, our modest dataset would implicate the creation of 53,000 files (or 26,500 if not considering symmetric similarities). It is clear that a meet-in-the-middle approach is the only way out.

In our implementation we use the concept of *file buckets* to balance the similarities output. We define a file bucket as a container of individual files. A file bucket itself is in fact again a file. Instead of creating a similarity file for every item, we spread out the similarities over the number of file buckets available. So a file bucket contains the similarities of 1 or more items depending on the number of file buckets used.

To decide which file bucket a similarity (e.g. (i_1, i_2)) should be written to, we assign a private numerical id to every available item in the system, ids ranging from 1 to n_i (the total number of items). Item similarities are then spread out over the file buckets using a modulo function. Table II shows how the similarities of our earlier example would be divided amongst 3 file buckets (ranging from 0 to 2).

We found that it was more efficient for the processing in the following phases to actually write every similarity to the file buckets instead of just the half triangle matrix. So for every calculation two values are written e.g. for $0_?$ we write both (i_1, i_2) to bucket 1 and (i_2, i_1) to bucket 2. Note that similarities will be evenly spread out over the

Table II
THE DIVIDING OF THE OUTPUTS IN FILE BUCKETS.

#	*item simil* (x, y)	file bucket
$0_?$	$(i_{\underline{1}}, i_2)$	1 ($\underline{1}$ *mod* 3)
$1_?$	(i_1, i_3)	1
$2_?$	(i_1, i_4)	1
$3_?$	(i_1, i_5)	1
$4_?$	$(i_{\underline{2}}, i_3)$	2 ($\underline{2}$ *mod* 3)
$5_?$	(i_2, i_4)	2
$6_?$	(i_2, i_5)	2
$7_?$	$(i_{\underline{3}}, i_4)$	0 ($\underline{3}$ *mod* 3)
...	...	...

available buckets but similarities of the same item (e.g. (i_1, i_2), (i_1, i_3), (i_1, i_4) and (i_1, i_5)) will always be in the same bucket. This allows for efficient loading of the similarities of an item (i.e. only one file bucket needs to be read), which will be required in the next phase.

Writing with a lot of different nodes (and cores) to the same file (bucket) can slow down the file system substantially. Therefore every core in every node should write to its own dedicated file buckets on its local disc. For every node the file buckets can then be merged locally first for the cores in that node, next over all nodes. Finally, the merged file buckets from all nodes and cores can then be stored on the shared storage. Fig. 3 visualizes this output process for 2 nodes with each 3 cores mapped on the HPC infrastructure.

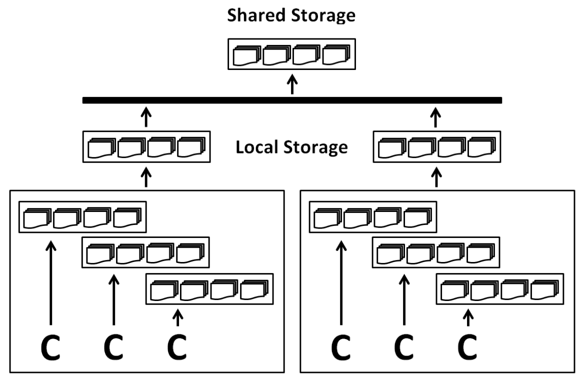


Figure 3. The merging file buckets strategy for two nodes with 3 cores. Every core writes to dedicated file buckets for optimal file system write efficiency. The file buckets are then merged per node and finally over all the nodes to the shared storage.

B. Phase 2: User Similarity

User similarity defines some degree of matching between two users. The term similarity here is somewhat misleading as we are interpreting the similarity of u_1 towards u_2 as the degree to which u_1 may provide interesting items for u_2 [7]. In contrast to the item similarity, the user similarity is not symmetric. The calculation of all the user similarities will therefore result in $n_u(n_u - 1)$ comparisons with n_u the number of users. Table III shows the user similarities for $n_u = 4$.

Table III
THE USER SIMILARITIES FOR 4 ITEMS.

	u_1	u_2	u_3	u_4
u_1	x	$0_?$	$1_?$	$2_?$
u_2	$3_?$	x	$4_?$	$5_?$
u_3	$6_?$	$7_?$	x	$8_?$
u_4	$9_?$	$10_?$	$11_?$	x

In our implementation, the calculation of the user similarity (u_1, u_2) requires as input the consumptions of both users and the item similarities of the items that u_1 consumed. If u_1 has rated for example two items i_x and i_y then the item similarities of i_x and i_y with every other item in the system must be loaded.

To load all the item similarities of a given item, we must simply load the corresponding file bucket determined by the private id of the item and the modulo function. Because of this file buckets structure there is no need to load all the item similarities except for in the worst case scenario where the needed items are spread out over all file buckets. This can of course be easily prevented by playing with the number of file buckets.

The loading of the consumptions should not pose any problems since even for big datasets like the movielens dataset with 10M ratings, the consumption file is but 262 MB and can easily be loaded in memory.

For optimization reasons, we parallelize the user similarity calculation tasks on a different granular level then we did for the item similarity. The calculation of the user similarity of (u_1, u_2) requires the item similarities of the items consumed by u_1 as does the (user) similarity calculation of (u_1, u_3) and (u_1, u_4). It therefore makes sense to load these item similarities once and then process (u_1, u_2), (u_1, u_3) and (u_1, u_4). In the situation of 4 nodes with each 3 cores available, we would divide the calculation tasks as follows.

$node_1$	$0_?$	$1_?$	$2_?$
$node_2$	$3_?$	$4_?$	$5_?$
$node_3$	$6_?$	$7_?$	$8_?$
$node_4$	$9_?$	$10_?$	$11_?$

So all the similarity calculation tasks of a user are handled by the same node, within the node the tasks can further be delegated towards the available cores.

The same output strategy as we proposed for the item similarities (Fig. 3) can be applied here. For efficiency reasons every core must again write to dedicated file buckets to be merged first locally on the node and then globally to the shared storage.

C. Phase 3: Recommendations

A recommendation is a match between an item and a user. To calculate the complete set of recommendations such a matching between every item and user must be made as shown in table IV.

Table IV
THE RECOMMENDATIONS FOR 5 ITEMS AND 4 USERS.

	u_1	u_2	u_3	u_4
i_1	.	.	.	.
i_2	.	.	.	.
i_3	.	.	.	.
i_4	.	.	.	.
i_5	.	.	.	.

Our recommendation algorithm requires for the recommendation of an item i to a user u both the user similarities of u and the item similarities of i. This maps directly onto the file buckets structure generated by the item and user similarity phases.

To match every user with every item, every generated file bucket in the item similarity phase must be matched with every file bucket from the user similarity phase (Fig. 4). This approach again allows easy scaling, since the couples of item and user file buckets can be divided amongst the available nodes (and cores) in the infrastructure.

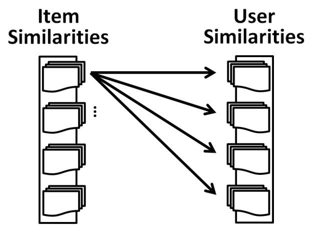


Figure 4. In the recommendation phase all the file buckets of the item and user similarities must be matched with each other.

A node needs only to load the item similarity file bucket and the user similarity file bucket as input for the recommendations of the items and users contained in the buckets. By increasing the number of file buckets, their file size can be reduced allowing a node to fully load the required data into RAM memory. A trade-off between the number of jobs (couples of buckets) and the size of the job (size of the buckets) will have to be made.

IV. RESULTS

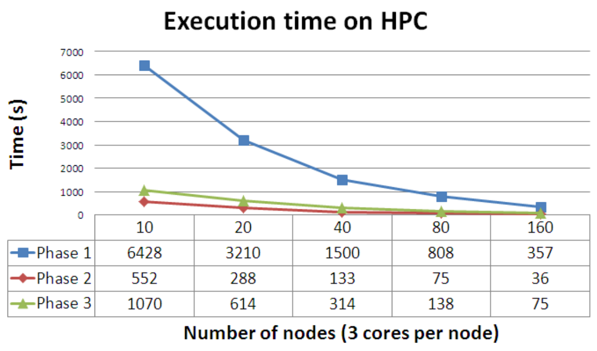
To validate our file-based recommendation approach we used a dataset collected from a popular cultural events website. For over 5 months we logged the explicit and implicit user feedback provided about the events by means of rating systems and page visits. This events dataset was particularly interesting to test our recommendation algorithm because events are *one-and-only* items [9] and difficult to recommend with a non-hybrid recommender.

The dataset contains the metadata of 53,000 items (events in this case) and consumptions of 1700 users. We aggregated the 14,000 consumptions (implicit and explicit) they produced, into 6800 consumptions by using a simple weighing scheme. In total 4700 unique events of the dataset were eventually consumed at least once.

Since the focus of this research is on the applicability of file-based approaches for recommendation systems, we will not present any quality metrics about the recommendations themselves. Instead we have plotted (Fig. 5) the execution time of the three introduced phases of our recommendation workflow.

By doubling the number of used nodes of the HPC infrastructure we can see the execution time of each of the phases decreases to halve the time. When repeating the experiment with a fixed amount of nodes but varying the number of cores in each node, similar results were obtained.

The execution time of the first phase differs significant towards the second and third phases. This is a result of the number of items versus the number of users available in the dataset (53,000 versus 1700) and therefore not a consequence of the employed algorithm. Note that in the third phase (as described in section III-C) the recommendation value for every item for every user in the system is calculated.



	10	20	40	80	160
■ Phase 1	6428	3210	1500	808	357
◆ Phase 2	552	288	133	75	36
▲ Phase 3	1070	614	314	138	75

Figure 5. The execution time of the three phases executed on the HPC infrastructure by 10, 20, 40, 80 and 160 computing nodes. All nodes having three processor cores available. For phase 1, 200 file buckets where used. Only 1 file bucket was used for phase 2 considering the small number of users.

V. Conclusion

We set out to see if we could free recommender systems from using databases without becoming irreversibly dependent on a custom NoSQL technology. We have managed to build a recommender system by using nothing more than a file-based data approach. The recommendation process was split up in three phases based on the input that a recommendation algorithm would require. We then showed for each phase how data could be structured in the form of file buckets so that computing nodes can simply read chunks of data into RAM memory and start processing independently of each other reducing the recommendation problem into an embarrassingly parallel computation problem. By deploying the recommender on a high-performance computing infrastructure and scaling the number of nodes put to the task, we found that our approach was both scalable and memory efficient.

VI. Future Work

Our future work will entail the validation of this work on larger datasets (like movielens and netflix) to find out the system's true potential and possible limitations. We also plan on implementing the recommendation algorithm on the Hadoop platform to see how it compares to our file-based approach in terms of performance.

Acknowledgment

Research funded by a PhD grant to Simon Dooms of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen). This work was carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules Foundation and the Flemish Government department EWI. We are grateful for technical support from the ICT Department of Ghent University.

References

[1] D. Lemire and S. McGrath, "Implementing a rating-based item-to-item recommender system in php/sql," Ondelette.com, Tech. Rep. D-01, January 2005.

[2] W. Woerndl, C. Schueller, and R. Wojtech, "A hybrid recommender system for context-aware recommendations of mobile applications," in *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop*, 2007, pp. 871–878.

[3] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," in *Proceedings of the fourth ACM conference on Recommender systems*, ser. RecSys '10, 2010, pp. 293–296.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.

[5] Z.-D. Zhao and M.-S. Shang, "User-based collaborative-filtering recommendation algorithms on hadoop," in *Knowledge Discovery and Data Mining, 2010. WKDD '10. Third International Conference on*, jan. 2010, pp. 478–481.

[6] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07, 2007, pp. 271–280.

[7] C. Cornelis, X. Guo, J. Lu, and G. Zhang, "A fuzzy relational approach to event recommendation," in *Proceedings of the Indian International Conference on Artificial Intelligence*, 2005.

[8] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol, "Data mining methods for recommender systems," in *Recommender Systems Handbook*, 2011, pp. 39–71.

[9] C. Cornelis, J. Lu, X. Guo, and G. Zhang, "One-and-only item recommendation with fuzzy logic techniques," *Information Sciences*, vol. 177, no. 22, pp. 4906–4921, 2007.