A MapReduce Implementation of the Spreading Activation Algorithm for Processing Large Knowledge Bases Based on Semantic Networks

Article · January 2012

DOI: 10.4018/jksr.2012100105

3 authors:

Jorge González Lorenzo
University of Oviedo
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Jose Emilio Labra Gayo
University of Oviedo
148 PUBLICATIONS **708** CITATIONS
SEE PROFILE

Jose Maria Alvarez-Rodríguez
University Carlos III de Madrid
60 PUBLICATIONS **266** CITATIONS
SEE PROFILE

A MapReduce implementation of the Spreading Activation algorithm for processing large knowledge bases based on semantic networks

Jorge González Lorenzo

Department of Computer Science, University of Oviedo, Spain

José Emilio Labra Gayo

Department of Computer Science, University of Oviedo, Spain

José María Álvarez Rodríguez

Department of Computer Science, University of Oviedo, Spain

Abstract

The emerging Web of Data as part of the Semantic Web initiative and the sheer mass of information now available make it possible the deployment of new services and applications based on the reuse of existing vocabularies and datasets. A huge amount of this information is published by governments and organizations using semantic web languages and formats such as RDF, implicit graph structures developed using W3C standard languages: RDF-Schema or OWL, but new flexible programming models to process and exploit this data are required. In that sense the use of algorithms such as Spreading Activation is growing in order to find relevant and related information in this new data realm. Nevertheless the efficient exploration of the large knowledge bases has not yet been resolved and that is why new paradigms are emerging to boost the definitive deployment of the Web of Data. This cornerstone is being addressed applying new programming models such as MapReduce in combination with old-fashioned techniques of Document and Information Retrieval. In this paper an implementation of the Spreading Activation technique based on the MapReduce programming model and the problems of applying this paradigm to graph-based structures are introduced. Finally, a concrete experiment with real data is presented to illustrate the algorithm performance and scalability.

Introduction

The Semantic Web (Berners-Lee, Hendler, & Lassila, 2001) is considered as an extension of the World Wide Web, adding metadata understandable by machines; and the same issues about size addressed in the traditional web are present. The Semantic Web data is expressed using RDF triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph. If data is modeled through a graph, then we can use graph algorithms to explore this data. One of these algorithms is the spreading activation algorithm (Todorova et al., 2009). But, if we take into account that the whole Semantic Web has billions of triples, it is obvious that the generated graph would be very large to fit into one single machine. Fortunately, this kind of size issues has been solved before in the traditional web by using parallel and distributed computing approaches. One of the most successfully applied framework for parallel and distributed processing is MapReduce (Dean, & Ghemawat, 2004). MapReduce has proven to be efficient resolving problems involving big data, whereas low latency is not required.

The main motivation of our research if providing a way of reducing the amount of time needed for processing large RDF datasets. In this article, an implementation of the spreading activation algorithm using the MapReduce programming model is presented. The spreading activation algorithm is used to find out related concepts starting from a set of activated nodes. The algorithm propagates this activation through the graph vertices, and, at the end, the related concepts are the ones with highest activation level. First, we describe the main problems of solving graph problems with MapReduce, and then the decisions and details about the final implementation of the algorithm are shown. Finally, a result section is presented to demonstrate the scalability of the implementation.

Previous work

MapReduce

MapReduce is a framework introduced by Google in 2004 for processing huge datasets using a large number of machines in a parallel and distributed way (Dean, & Ghemawat, 2004). MapReduce framework transparently handles system-level details, such as scheduling, fault tolerance or synchronization. The main advantages of the framework is the simplicity of the map and reduce operations, that allow a high degree of parallelism with little overhead, at the cost of writing programs in a way that fits this programming model. MapReduce has proven to be efficient and is used by Google internally for processing petabyte order datasets. This success has motivated the apparition of the open source initiative Hadoop[1], which is an Apache project mainly developed and supported by Yahoo.

MapReduce handles all the information using tuples of the form <key, value>. Every job consists of two phases: a map phase and a reduce phase. The map phase process the input tuples and produce some others intermediate tuples. Input tuples are divided in groups, each of them processed by a map function running in a single machine. Then, these intermediate tuples are grouped together according to their key value forming a group. Finally, each group is processed by the reduce function, producing a set of output tuples.

Spreading Activation

The spreading activation algorithm (hereafter SA) was introduced as an approach for modeling the human memory and its cognitive processes by following its low-level structure (Todorova et al., 2009). It takes advantage of the similarity between neural networks and graph models, so the same ideas behind the spreading activations mechanisms can be applied to graphs modeled problems. The algorithm has been successfully applied to problems like categorization, information retrieval and search engine ranking (Álvarez et al., 2011; Crestani, 1997; Troussov et al., 2008; Ziegler & Lausen, 2009; Berrueta, Labra, & Polo, 2006). Moreover these kind of

[1] http://hadoop.apache.org

techniques and technologies (Lytras & Pablos, 2011) are being applied to different sectors in order to promote the information society (Sharma, Dharmawirya, & Samuel, 2010*)* in areas such as e-Health, e-Procurement or e-Government.

The SA process starts with the activation of an initial set of vertices of the graph. An initial activation value is assigned to these vertices and then this activation is propagated to the connected vertices, taking into account the weight of the edges that join these vertices. This operation is repeated in an iterative process which uses a decay factor to model the idea of the energy of the activation dying out. The process stops after a predetermined number of iterations. Nodes with an activation level over a certain threshold are considered active. The inactive nodes are filtered out and the selection ends up with a sub-graph representing the nodes most related to the initial ones.

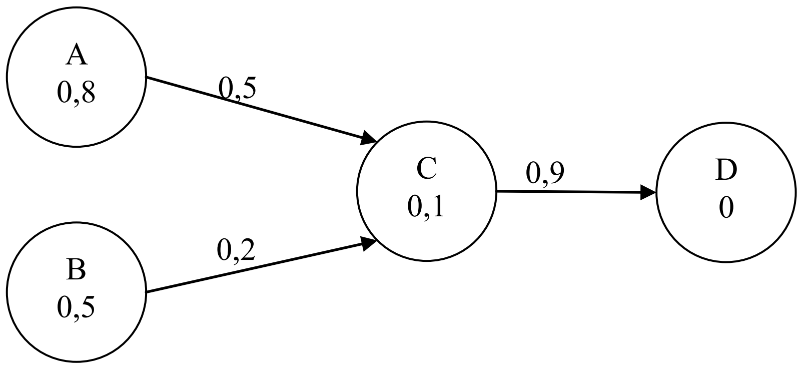

Figure 1. Spreading activation graph example

The illustration (Figure 1) shows a basic example of semantic graph with four vertices (A, B, C and D) and three edges connecting them. Vertices have an activation level and edges have a weight. At a given time, activation levels are the one shown in Figure. 1, and we set a firing threshold of 0,4. In order to calculate activation levels for next iteration, we should propagate the activation of vertex A and B to vertex C, given that A and B are the only vertices with activation level over the threshold, using the formula:

$$A_j = A_j + \sum_i A_i \cdot w_{ij} \cdot D$$

Activation level of vertex C is calculated as the sum of incoming-connected vertices activation (A and B in the example), considering edge weight and a decay factor D. In the next iteration, vertex C is supposed to have activation over the threshold so it will propagate activation to vertex D, and so on.

Spreading Activation and MapReduce

Graph problems and MapReduce

In order to apply MapReduce to solve a SA problem we need to take into account some considerations. Since the SA algorithm resolution is ultimately a graph problem, we will need a MapReduce algorithm that also operates on a graph. This will bring some disadvantages. Due to the large size of the datasets, this graph is also expected to be very large, and therefore impossible to fit into one single machine. For example, a FOAF graph can span millions of vertices and edges, making very difficult to analyze it. Representing the graph as a matrix and trying to solve it using matrix operations is neither a valid solution for the same reason, matrix would be too big. We need to divide data and parallelize calculations in order to be processed independently using many machines at the same time. The key idea for this implementation is to consider the graph as a group of smaller graphs each of which can be processed individually in a single node, not requiring data from other nodes. The RDF data graph can be represented as a set of connections, each one comprising two nodes (subject and object) and an edge (predicate). Each machine of the cluster will have only a small set of connections and perform operations on this piece of data knowing nothing about other parts of the graph. Using this approach we solve another problem: the input is not ordered. For a given vertex, some ingoing or outgoing edges will be found at some point of a file, and some other edges of the same vertex will be in other point. With this approach we don't need to have all the edges of a node at the same time in order to make calculations. But if we need so, the MapReduce framework will do this for us in the shuffle phase. In other words, for a given vertex, calculations that don't need all the information of that vertex at the same time will be done in the mapper function, while calculations that need all the information about a vertex will be performed in the reducer function. This approach has proved to be successful in similar algorithms such as Pagerank (Lin, & Dyer, 2009).

Applying MapReduce to the Spreading Activation algorithm

The SA resolution can be seen as an iterative process. At a given time we have some vertices with an activation level over a certain threshold, others below the threshold and others not activated at all. Vertices activated over the threshold propagate activation to the outgoing edges. Therefore, vertices receiving activation need to know all the incoming values of their incoming edges in order to calculate their final activation value for a given iteration. So we can identify two phases here. The first one iterating on all the vertices of the graph, checking if activation is over the threshold and, if so, propagating activation to vertices connected by outgoing edges. This phase can be performed independently on each vertex of the graph not needing any other information, so we will carry out these calculations in the mapper function. The other phase would be, for each node, sum all the incoming activations in order to calculate the final activation of the vertex at the end of this step. This will require grouping all the data involving a given vertex; fortunately the MapReduce framework will do this for us. MapReduce group the pairs before the reduce phase attending to the key value. In the map phase, for a given vertex, if we emit pairs pointing out in the key the identifier of the outgoing-connected vertex and in the value the activation provided by the current vertex, the framework will group all the ingoing activation values for a given node in the reducer function. Then, we will only need to sum these values inside the reducer function. At this point, current iteration would be finished, and pairs emitted in the reduce phase are used as input in next iteration. This iterative process can finish on different conditions, like a fixed number of iterations.

Iterative graph problems and MapReduce

There is a disadvantage on using MapReduce for solving iterative graph problems: graph structure must be available at the end of each iteration. In our case, if we just emit pairs of activated vertices, all the other ones will be missed in the next iteration. This issue is commonly resolved passing along the graph structure from the mappers to the reducers (Lin, & Schatz, 2010). In other words: the mapper do normal algorithm calculations, emits a pair for every edge of an activated vertex, but also emits a pair for every node and its connections, no matters if that node is activated or not. We will denote these vertices as network vertices. Note that this is already done at the end of the mapper function. A special flag indicating if the pair is a network structure pair is necessary and is used in the reducer to distinguish between network pairs and activation pairs. The reducer combines, not only all the incoming activated edges but the network structure pairs in order to obtain at the end of the iteration the whole graph again. In other words, for a given vertex, network pairs provide the outgoing edges of the vertex, and not network pairs are used to calculate the activation level of that vertex.

System Architecture

We divide the full processing in three stages. The pre-processing phase imports data and set initial values. Then the spreading phase computes activation and spread it over the graph iteratively. Finally the post-processing phase retrieves relevant information from the output. Each of these stages is implemented as a MapReduce job.

Pre-processing

In this stage, initial activation levels of vertices and weights of edges are set to a specific value. This phase is executed only once, before the spreading phase. The number of edges and vertices in the RDF graph can be very large, therefore it can be interesting the use of methods like default values or regular expressions, and specify individual activations and weights only for a small set of elements.

Spreading: Mapper function

The mapper function receives pairs of the form <key, value> where key is the vertex identifier, and value is a structure that contains the current activation value of the vertex, and a list of outgoing edges, one for each outgoing connection with other vertex. This connection info will be composed of a vertex identifier and a weight. The mapper function then checks, for the current pair, if the vertex is activated and if so, spread the activation to the connected vertices. In order to spread the activation, it has to iterate over all the vertex outgoing connections, and emit a pair where key is the identifier of the outgoing vertex, and the value is the source vertex activation value multiplied by the edge weight and a decay factor.

Map function pseudo code

```
map(key, value)
  if value.activation > threshold and not value.visited
    value.visited = true
    for connection in value.connections
      output.activation = value.activation *
        connection.weight * decay_factor
      output.network = false
      emit(connection.node_id, output)
    end
  end
  value.network = true
  emit(key, value)
end
```

Following the example illustrated in Figure 1, the only vertices activated are A and B, they both have vertex C as destination. Therefore two pairs would be emitted <C, 0.4> and <C, 0.1>.

Key value is C in both cases, corresponding to the destination vertex of A and B, and the float value is vertex activation level multiplied by edge weight. A pair for network reconstruction is also emitted, one for each edge of the graph. The need of these pairs is explained in the section “Iterative Graphs Problems and MapReduce”.

Spreading: Reducer function

The reducer function receives a key and a tuple of values, where key is the vertex identifier and each value is a structure containing an activation value provided by each incoming-connected vertex. Previously, the framework has grouped all the pairs with same value, so in each call to the reduce function we have all the incoming activation levels for a given node. In order to calculate the total activation of this vertex, all activations values must be added together. Finally, a pair with the node identifier used as key, and a value structure with the total activation level and connections is emitted. It is also necessary to distinguish between normal vertex info and network info, the first ones contain only activation values the second ones contain also connection information.

Reduce function pseudo code

```
reduce(key, iterator values)
  total_activation = 0
  network_activation = 0
  visited = false
  for node_info in values
    if not node_info.network
      output.activation += node_info.activation
    else
      output.connections += node_info.connections
      network_activation = node_info.activation
      visited = node_info.visited
    end
  end
  if not visited
    output.activation = network_activation
  end
emit(key, output)
end
```

Following with the map example, we have pairs <C, 0.4> and <C, 0.1> that had been emitted before. Now, the framework would group these two pairs, since both have the same key. So these two pairs will be processed in the same reduce call. A pair with key C and a list of activations as value would be received <C, [0.4, 0.1]>.

The reducer function should now sum all the activations values in order to calculate the final activation of C at the end of this iteration. Edges going out from C must also be emitted as value in order to have available the network topology in next iteration.

These special pairs are emitted by the map function, and combined here in the reduce with the activation information (see section “Iterative Graphs Problems and MapReduce” for further details). The final emitted pair would be <C, [0.5 D]>, containing the node identifier as key, and the final activation and reachable vertices, in this case 0.5 and D respectively.

Post-processing

This phase consists in retrieving results from the output. Since the algorithm is an iterative process, the results are the output of the last iteration. This task can be different depending on the kind of problem we are solving and the results we want to extract. Usually, output is so large that another MapReduce job is advisable for parsing the data. Some common tasks in this phase are ordering and normalization.

Results

For the evaluation of our implementation we have set up a cluster comprising 16 nodes and a 10/100 MB connection. Each node is equipped with a Pentium 4 dual core processor, 1GB RAM and a 160GB hard disk. We have used the open source implementation Apache Hadoop, version 0.20.203, latest stable version to date.

The experiment program consists of a configuration of the algorithm for finding related people using FOAF data. The chosen dataset is the one used in the Billion Triple Change [2]in the year 2009, which is real-world data crawled from many different semantic search engines. We set up a high activation value to one or more specific individuals for those who we want to find out related people, and weighting the FOAF: knows relation over the rest. Then, we run the program on the cluster and, after the execution finishes, we obtain a sorted list of related people. In order to make calculations over the whole input and avoiding discarding most of the RDF graph, we set up a default activation value and a default weight for every vertex and edge in the net, so every vertex is processed and all the connections are used as a part of the calculations. We configured the algorithm to execute four iterations.

We made two different experiments using this program, in order to obtain two different measures. The first one for proving algorithm scalability as data grows, and the second one to measure how execution time behaves when more nodes are added to the cluster. Every test has been repeated three times in order to obtain average measures.

We report in Figure 2 the result of the first experiment varying input data size and maintaining the number of nodes in the cluster to a fixed value. For this experiment we have used 16 nodes in the cluster.

[2] http://challenge.semanticweb.org

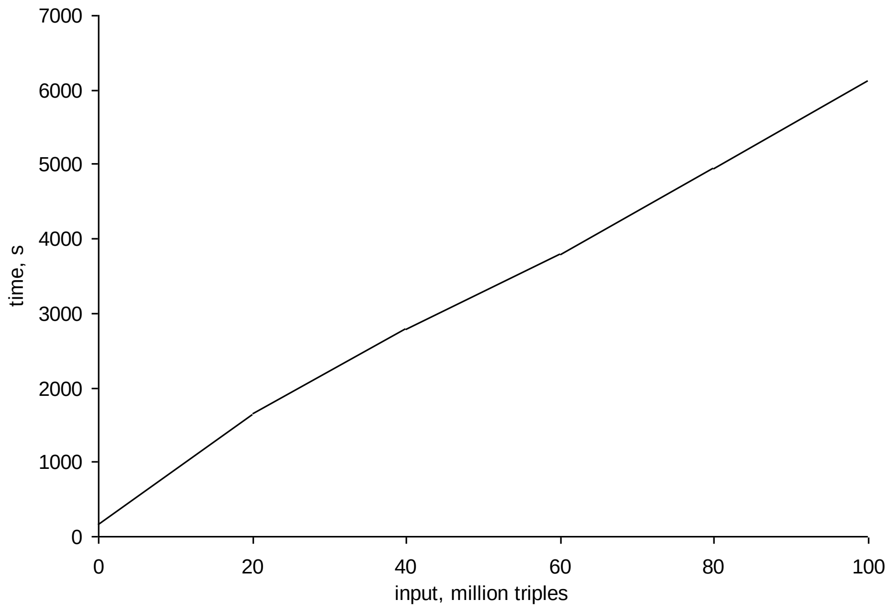

Figure. 2. Scalability with different input size

A linear correspondence between execution time and input size can be observed, proving that the algorithm scales well as data grows.

The other experiment keeps a fixed input data size and changed the amount of nodes in the cluster. The same program for finding related people, with exactly the same configuration has also been used for this experiment. We started with a single node configuration, and repeated the experiment with 2, 4, 8 and 16 nodes. Results of this experiment are shown in Figure 3.

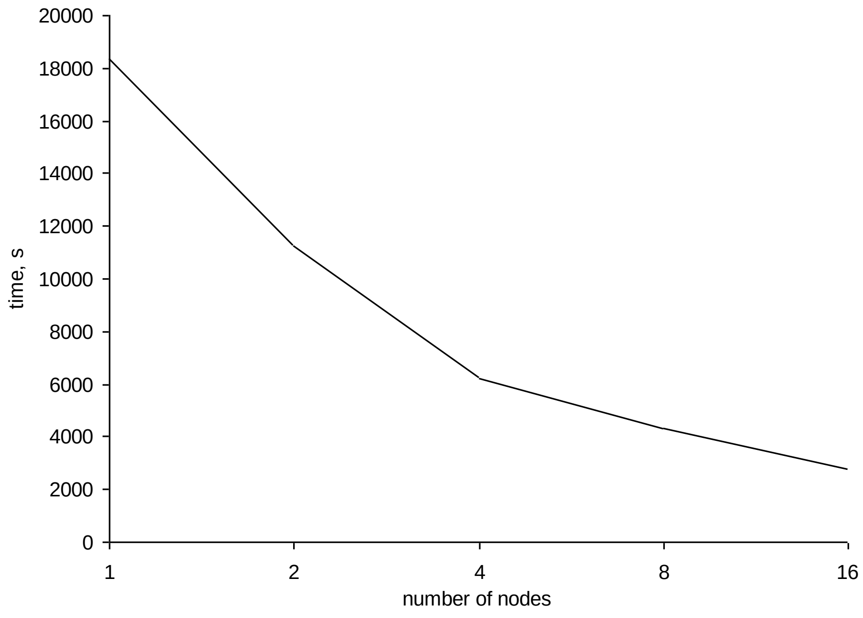

Figure 3. Scalability with different nodes

In Figure 3 we can observe how execution time reduces as more nodes are added to the cluster. This is the expected result and means that the algorithm scales well as more computing units are used, something essential for a MapReduce program. However, performance does not increase linearly as more nodes are added. If we take a look at execution time in Figure 3, we can see that there is not a linear correspondence between time and number of nodes. This is because of the overhead introduced by the framework. A fragment of time is lost in setting up machines and communication between the different nodes in the cluster. If too many nodes are used and input is not big enough, then overhead time becomes significant compared with computation time. We can conclude that, for a fixed input size, there is an amount of nodes above which is not worth adding more. Also, there are other framework configuration parameters, such as splits size that can dramatically affect performance, as has been stated by (Kambatla, Pathak, & Pucha, 2009).

Related work

Jimmy Lin and Chris Dyer (2008) have studied the application of MapReduce to text processing problems in the context of natural language processing, information retrieval and machine learning. More specifically, they have used MapReduce on graph problems, like the calculation of Pagerank. They have also stated common mistakes, proposed design patterns and strategies for solving graph problems using MapReduce. Some of the issues presented in their research were found during the development of our implementation, especially graph structure problems.

Schätzle, Przyjaciel-Zablocki and Lausen (2011) and Urbani, Maaseen and Bal (2010) have applied MapReduce to many different problems related to the Semantic Web and RDF graphs. They have worked on mapping SPARQL to Pig Latin, and done some research on processing large RDF graphs using MapReduce. Their work is focused on performing path queries on very large graphs.

In recent years and regarding the emerging use of ontologies in the Semantic Web area new applications of SA have appeared to explore concepts (Qiu & Frei, 1993; H. Chen & Ng, 1995; Teufl & Lackner, 2011). Dix et al. (2010) describe a collection of methods to allow SA to be used on web-scale information resources. In their work, they use custom methods and procedures to manage semantic data, with special emphasis on RDF repositories and data caching. Álvarez et al. (2011) and Kreuzthaler et al. (2011) have applied successfully the Spreading Activation algorithm over medical ontologies for recommending concepts and some applications to collaborative filtering (Pham, Cao, Klamma, & Jarke, 2011) (community detection based on tag recommendations, expertise location, etc.) are being considered to apply the SA technique.

On the other hand, Jacopo Urbani, Kotoulas et al. (2010) have developed a distributed reasoner using MapReduce that works on web scale input data. They describe the problem of very large scale reasoning and propose an Apache Hadoop implementation that outperforms all existing solutions.

Conclusions and future work

Due to the enormous quantities of data generated in the Semantic Web, parallel and distributed computing paradigms are more crucial than ever. Our purpose was to take advantage of the widely used framework MapReduce and apply it in the Semantic Web context. We have introduced an implementation of the spreading activation algorithm, widely used on RDF datasets, applying the MapReduce programming paradigm. Algorithm scalability has been evaluated on a 16 nodes cluster using real-world data. This implementation has proved to manage large amounts of data and scale out by using more machines simultaneously. It has also proved to scale linearly as data grows. Future research will attempt to find specific uses of the algorithm on large datasets.

Jacopo Urbani, Maaseen and Bal (2010) have proved that dictionary encoding can improve overall performance of text processing MapReduce jobs despite the overhead that it introduces. In their work, they prove that a MapReduce algorithm can efficiently compress and decompress large amounts of data, and also scale linearly as input grows. Dictionary encoding is a technique that would probably improve our SA implementation, avoiding passing along large URI strings.

References

Álvarez, Jose María, Polo, Luis, Abella, Pablo, Jiménez, Weena, Labra, Jose Emilio, (2011). *Application of the Spreading Activation Technique for Recommending Concepts of well-known ontologies in Medical Systems.* In Proc. of SATBI collocated with ACM-BCB 2011. Chicago, USA.

Berners-Lee, T, Hendler, J. and Lassila, O. (2001). *The Semantic Web, Scientific American*, 284(5): (p. 34-43).

Berrueta, Diego, Labra Gayo, José Emilio, Polo, Luis, (2006). *Searching over Public Administration Legal Documents Using Ontologies, In Proc. of Joint Conference on Knowledge-Based Software Engineering.*

Crestani, F, (1997). *Application of Spreading Activation Techniques in Information Retrieval, Artificial Intelligence Review, 11*, 453-482.

Dean, J. and Ghemawat, S., (2004). *Mapreduce: Simplified data processing on large clusters, In Proc. of the USENIX Symposium on Operating Systems Design & Implementation.* OSDI.

Dix, A., Katifori, A,. Lepouras, G., Vassilakis, C. and Shabir, N. (2010). *Spreading Activation Over Ontology-Based Resources: From Personal Context To Web Scale Reasoning, International Journal of Semantic Computing.*

Kambatla, Karthik, Pathak, Abhinav and Pucha, Himabindu, (2009). *Towards Optimizing Hadoop Provisioning in the Cloud, In Proceedings of the conference on Hot topics in cloud computing.* HotCloud.

Kreuzthaler, M., Bloice, M. D., Faulstich, L., Simonic, K.-M., & Holzinger, A. (2011, apr*). A comparison of different retrieval strategies working on medical free texts*. International Journal of Universal Computer Science (JUCS) , 17 (7), (p. 1109-1133).

Lin, Jimmy, Dyer, Chris, (2009). *Data-intensive text processing with MapReduce, In Proc. of Human Language Technologies*.

Lin, Jimmy, Schatz, Michael, (2010). *Design Patterns for Efficient Graph Algorithms in MapReduce, In Proc. of the Eighth Workshop on Mining and Learning with Graphs.* MLG.

Lytras, M., & Pablos, P. O. de. (2011, may). *Software Technologies in Knowledge Society*., International Journal of Universal Computer Science (JUCS), 17 (9), (p. .1219-1221).

Pham, M. C., Cao, Y., Klamma, R., & Jarke, M. (2011, feb). *A clustering approach for collaborative filtering recommendation using social network analysis*. International Journal of Universal Computer Science (JUCS), 17 (4), (p. 583-604).

Przyjaciel-Zablocki, Martin, Schätzle, Alexander, Hornung, Thomas, Lausen, Georg, (2011). *RDFPath: Path Query Processing on Large RDF Graphs with MapReduce,* 1st Workshop on High-Performance Computing for the Semantic Web. HPCSW.

Schätzle, Alexander, Przyjaciel-Zablocki, Martin, Lausen, Georg (2011). *PigSPARQL: Mapping SPARQL to Pig Latin*, In Proceedings of 3th International Workshop on Semantic Web Information Management SWIM.

Sharma, R. S., Ng, E. W., Dharmawirya, M., & Samuel, E. M. (2010). *A Policy Framework for Developing Knowledge Societies.* International Journal of Knowledge Society Research (IJKSR), 1(1), 22-45. doi:10.4018/jksr.2010010103

Teufl, P., & Lackner, G. (2011, apr). *Knowledge extraction from RDF data with activation patterns*. International Journal of Universal Computer Science (JUCS) , 17 (7), (p. 983-1004).

Todorova, P., Kiryakov, A., Ognyano, D., Peikov, I., Velkov, R., & Tashev, Z. (2009). *D2.4.1 Spreading Activation Components (v1) (Tech. Rep.)*. LarKC FP7. Project 215535.

Troussov, A., Sogrin, M., Judge, J., and Botvich, D, *Mining sociosemantic networks using spreading activation technique, In Proc. of IMEDIA' 08 and I-KNOW' 08. JUCS*, 2008.

Urbani, J., Maaseen J., & Bal, H. (2010). *Massive Semantic Web data compression with MapReduce, In Proc. of the MapReduce workshop at HPDC*.

Urbani, Jacopo, Kotoulas, Spyros, Maassen, Jason, Drost, Niels, Seinstra, Frank, Van Harmelen, Frank, Bal, Henri, (2010). *WebPie: A Web-Scale Parallel Inference Engine, In Proc. of the third IEEE International Scalable Computing Challenge.* SCALE.

Ziegler, C.-N. and Lausen, G., (2004). *Spreading Activation Models for Trust Propagation, In Proc. of the IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE '04),* Taipei, March.