Beyond Traditional NLP: A Distributed Solution for Optimizing Chat Processing
Automatic Chat Assessment using Tagged Latent Semantic Analysis

Mihai Dascălu, Ciprian Dobre, Ştefan Trăuşan-Matu, Valentin Cristea
Computer Science Department
"Politehnica" University of Bucharest
Bucharest, Romania
mihai.dascalu@cs.pub.ro, ciprian.dobre@cs.pub.ro, stefan.trausan@cs.pub.ro, valentin.cristea@cs.pub.ro

Abstract—With the increasing popularity and evolution of Computer Supported Collaborative Learning systems, the need for developing a tool that automatically assesses instant messaging conversations has become imperative. The main reasons are the high volume of data and the increased amount of time spent for manually assessing conversations. We propose an automated analysis system based on Natural Language Processing (centered on Latent Semantic Analysis and Social Network Analysis) and optimize its runtime performance by means of distributed computing. Moreover, we provide a unique grading mechanism based on a multilayered architecture and induce an increase of speedup by deploying a Replicated Workers architecture. Load balancing and fault tolerance represent key aspects of this approach, besides the actual increase in performance.

Computer-Supported Collaborative Learning; Automatic Chat Assessment; Tagged Latent Semantic Analysis; Replicated Workers Paradigm; Fault Tolerance

I. INTRODUCTION

Collaborative applications on the web were permanently developed in the last years in many domains. One remarkable case is their usage for educational purposes. Computer Supported Collaborative Learning (CSCL) using instant messenger and forums is very well suited from both practical and theoretical reasons. Chats and forums are commonly used by students and, moreover, they offer the possibility of joint learning anytime and anywhere. CSCL is also based on a different paradigm, which views learning as being a participant to a dialogue that constructs a discourse rather than a transfer of knowledge from teachers or textual documentation [1, 18].

Manual assessment of CSCL chats and forums is a very time consuming task. It is very difficult to follow the logs of conversations and to remember what every participant said and how their contributions are linked in a coherent discourse [18]. Very few automated CSCL chat analysis systems were developed, the cause being mainly due to the limits of Natural Language Processing (NLP) [5, 18]. The system [18, 19] is based on a polyphonic model following the participation paradigm. It is implemented with NLP (mainly Latent Semantic Analysis – LSA) and Social Network Analysis (SNA) techniques. Because one of the problems of NLP applications is the very high amount of time needed by some algorithms, distributed processing solutions may bring huge improvements.

We are in an era where distributed computing is a commodity. Distributed infrastructures provide a wide spectrum of benefits: transparent access to and better utilization of resources, increased computing and storage capacities, flexibility, adaptability and automation through dynamic and concerted interoperation of networked resources, cost reduction through various utility models, higher quality of products designed and developed via distributed tools, shorter time-to-market, and many more. This revolution is already well underway in scientific and engineering organizations, with high demand of computing and data processing.

With regards to data processing we are, as some authors put it, in an era of “Big Data” [8]. Many enterprises collect large datasets that record customer interactions, product sales, results from advertising campaigns on the Web, etc. Facebook alone collects 15 TeraBytes of data each day into its PetaByte-scale data warehouse [9]. Powerful telescopes in astronomy, particle accelerators in physics, and genome sequencers in biology are putting massive volumes of data into the hands of scientists. The ability to perform scalable and timely analytical processing of these datasets to extract useful information is now a critical ingredient for success. Still, cost-effective processing of large datasets is a nontrivial undertaking.

Parallel data-flow systems such as Map-Reduce and Hadoop have recently experienced a surge in popularity [15]. These systems are increasingly used for data warehousing and analytics, either directly or through the use of a high-level query language that is compiled down to a parallel dataflow graph for execution [3]. In this context we too present a solution that optimizes chat processing using Map-Reduce. Our solution, however, is different because we are optimizing an application that is both data and process-intensive. In the same time, the chat processing solutions suffers, as described in the architectural overview, from limitations regarding possible decompositions of tasks due to data synchronization and consistency issues. In this we present the mechanism to achieve results that, confirmed by the evaluation experiment, lead to performance far better than anticipated regarding the actual runtime; fault tolerance at worker level and robustness of the architecture are also major improvements compared to the initial sequential evaluation of the corpus. From this point of view one can say the presented approach represents also a methodology that developers can use to support the optimization through distribution of tasks for other related applications as the one we present.
In this paper, we present an implemented system used for evaluating chat conversations that relies primarily on LSA, but also employs other NLP techniques (part of speech tagging, parsing, speech acts identification, implicit links identification by means of repetitions, co-references and predefined patterns) and SNA [18, 19]. Thus, section 2 addresses the process of evaluating a single discussion. Section 3 represents the focus of this paper and presents the system’s architecture for evaluating a corpus of chats in a distributed manner. The paper ends with the key results from a validation session, conclusions and future improvements.

II. THE EVALUATION PROCESS OF A SINGLE CHAT

The scope of the system is to provide feedback and support to students that are using chats as part of their learning process. It also helps tutors with feedback regarding the involvement of their students in the discussions.

Because of the complexity and the holistic perspective seen as objective for automatically evaluating conversations, our approach combines natural language processing with Latent Semantic Analysis [4], Social Network Analysis [10], plus surface analysis metrics derived from Page’s essay grading studies [11]. The following part of this section provides more insight in the technologies used within our system. A more detailed description can be found in [17] and [18].

A. Architecture and Technologies

Although the method for chat processing has been described in detail in [17], key aspects of the evaluation are presented to better emphasize the need for an optimization using distributed processing. For providing relevant feedback to students and tutors, the system uses a multilayered architecture (depicted in Figure 1).

![Figure 1. Multilayered analysis architecture.](image)

The first steps in analyzing a discussion consider surface metrics derived from Page’s essay grading techniques [11] and readability measures [12]. These perspectives provide the basis for the quantitative evaluation of a user’s involvement and his degree of knowledge in general, only by evaluating the discussion at a lexical level.

The qualitative dimension is ensured by integrating a semantic approach modeled by means of LSA. But in order to reach the third layer, basic natural language processing techniques consisting of stop words elimination, spellchecking, stemming, tokenizing, part of speech tagging and parsing need to be addressed.

Moreover, discourse analysis is performed by applying techniques such as speech acts identification, lexical chains, adjacency pairs and co-references. These techniques are used for discovering a deeper structure of the discourse and for identifying interactions among participants [18].

The utterance graph is built starting from the previously identified implicit references, by enriching them with explicit links defined by participants in their collaborative chat environment [17]. This structure plays a central role in the marking process of each utterance and of each participant [16].

The most important component of the system is the latent semantic space used for estimating the similarity between concepts and utterances. LSA is a technique used in vector-space based semantics for analyzing relationships between a set of documents and the contained terms, by indirectly correlating them through concepts [4]. This transformation is obtained by a singular-value decomposition of the array and a reduction of its dimensionality. In our experiments we considered a projection over 300 dimensions that is considered an optimal empiric value by multiple sources [7].

This space is used for evaluating the cohesion of each unit of analysis (utterance) to its corresponding threads, discourse coherence and the overall importance and relevance of each utterance with regards to the entire discussion [19].

Besides applying term frequency - inverse document frequency on the initial term-document matrix build from a corpus of chats), tagging and segmentation are also addressed.: POS tagging is applied on all remaining words after stop words elimination and spellchecking are performed. Moreover, verbs are reduced to their corresponding stems for reducing the complexity of the generated space. According to [6] and [7], stemming applied on all words reduces overall performance because each form expresses and is related to different concepts.

Social network analysis considers the social graph induced by the participants and their corresponding interactions. SNA specific metrics are also applied on the utterance graph for identifying the central utterances within each discussion thread ([16] and [19]). Therefore, SNA is used at the surface level for modeling the interaction between participants as number of interchanged utterances, but also at the semantic layer by taking into consideration the mark of each utterance determined by means of LSA [16].

The overall grading process combines the results from all previous layers in order to provide a grade proposal for each participant.

III. CORPUS ASSESSMENT

A. General presentation and motivation of our approach

The assessment process of a single chat is a time consuming process. In average it takes from 3 to 10 minutes per chat, depending on the actual size of the chat – from approximately 150 utterances to 450 utterances. This is mostly
due to the initial NLP pipe processing. The initial evaluation experiments were performed sequentially using dual-core CPUs running at 2.13 GHz with 3MB cache and 2GB of RAM (a maxim of 1GB was allowed for data-structures stored in Heap space).

As previously described, among the involved sub-components, Part of Speech Tagging takes most time to evaluate. Moreover, this component can be considered a bottleneck for parallel computations because multiple threads have to access the static method offered by Stanford Log-linear Part-Of-Speech Tagger [13] in a synchronized manner to avoid concurrency issues.

On the other hand, implementing a multithreaded solution on the same physical machine has limitations with regards to available resources. The evaluation process is highly computational, with little I/O input. For example, at start-up, over 100MB of RAM are used for loading the initial configuration and predefined matrices. This step is independent from the actual length of the analyzed conversation because all configurations and the entire semantic space are loaded at startup. Afterwards, due to the internal structures used within the evaluation process, the allocated memory expands to 1GB of RAM in most cases. Moreover, all the graphs generated by the probabilistic parser in its attempt to find the best corresponding part of speech are not freed immediately from the memory. The Garbage Collector is responsible for cleaning up unused space and is triggered, in most cases, by the lack of free Heap space. This explains the general behavior of rapidly expanding to the maximum threshold of memory and bouncing below the upper bound.

In order to obtain a clear perspective over the actual assessment, the size of the input and output files must be also taken into consideration: an initial chat input file is an XML file ranging from 50KB to 150KB while, as output, the system offers serialized objects ranging from 1.8MB to 25MB.

Considering that a multi-threaded, single machine solution is not satisfactory, a new approach had to be promoted, enabling the assessment a corpus of chats with regards to:

- Parallel computation of chats on different machines;
- Load balancing of task assignments;
- Failover capabilities with the possibility of reassigning failed tasks until the entire corpus is assessed.

The promoted solution is based on the Replicated Worker Paradigm because tasks can be dynamically created as they are generated during the execution of the master/coordinator process. The evaluation processes are identical on each machine and are represented by replicated workers assigned to run on separate physical processors. In addition, the proposed parallel decomposition of the processing operations is also fault-tolerant. We propose a solution that detects when tasks fail to execute and that is capable of re-submitting them on other working nodes. Thus we optimize the chat processing tasks not only by distributing the actual computation on multiple nodes, but also by continuously monitoring and controlling their execution in order to provide a best-effort approach.

The solution uses a set of distributed work pools as shown in Figure 2. These work pools are used for controlling the allocation of tasks to the corresponding workers. A work-pool is a collection of tasks waiting to be executed by a single worker. When creating a new worker, it registers to the master process and awaits corresponding tasks to be assigned. When it becomes idle, it sends the results to the master and retrieves a new task from its work-pool. After all tasks in the input folder are completed, the master signals all workers for terminations and finishes its own execution.

Message queues are used for communication, providing an event driven approach and the possibility to send serialized objects between the producer and the consumer of the message. A general message status queue for all workers is used for signaling purposes. Task assignments are performed using dedicated queues for each slave. In our current implementation, this process is based on the Apache’s ActiveMQ message broker [14].

The master process also ensures load balancing by assigning new tasks to workers as soon as the current one is finished. Therefore the strategy used for planning tasks is First Come First Served (FCFS).

Failover and redundancy at the worker level are achieved by implementing a modified KAMA – Kaufman Adaptive Moving Average – algorithm. The original method, created by Perry Kaufman [2], considers the noise of the market. The approach is based on following the general trend and continuously adapting the prediction such that if changes are small and noise is marginal, the predicted values follow the original values very closely. On the other hand, if there are high fluctuations in the received values, KAMA follows with larger distance to lessen the number of false predictions.

From the market prices to failover and redundancy, the step is performed using a keepalive thread on each worker which signals its activity periodically and a monitoring service on the master that evaluates whether a worker is still running. If a worker becomes overloaded, the time between two keepalive messages will increase and the master process must predict correspondingly the next value.

![Figure 2. Replicated Workers Architecture](image-url)
SC (Smoothing Constant) is a standard part of the moving average construction and it determines the level to which the moving average is sensitive to current value swings. SC ranges from 0 to 1: the lower the Smoothing Constant, the less sensible the moving average is. In this way SC follows not only direction, but also volatility of values.

The assumes parameters are defined as:

- A (Actual values) represent the real time of a received keep-alive message;
- P (Predicted values);
- n is the Window size or analysis period; in our current implementation the selected value for this parameter is 5;
- α is the Smoothing constant; it is initialized with $\alpha = \frac{2}{n+1}$.

Starting from these definitions, the steps performed in the monitoring process are:

1) Calculate the ER (Efficiency Ratio) as direction of heartbeat messages divided by volatility. The more the fluctuation, the greater the rate; on the other hand, if values are constant in time, the direction is 0 and ER becomes also 0:

$$\text{Direction} = A_i - A_{i-n}.$$  \hspace{1cm} (1)

$$\text{Volatility} = \sum_{i=n}^{i-1} |A_{i-n} - A_{i-n+1}|.$$ \hspace{1cm} (2)

$$\text{ER} = \frac{\text{Direction}}{\text{Volatility}}.$$ \hspace{1cm} (3)

2) Using the shortest and longest moving average, we determine the SC of these averages. The used window sizes range from 2 messages to 10 messages:

$$SC = \text{ER} \times (\text{Fast}_{SC} - \text{Slow}_{SC}) + \text{Slow}_{SC}.$$ \hspace{1cm} (4)

where

$$\text{Fast}_{SC} = \frac{2}{2+1}, \text{Slow}_{SC} = \frac{2}{10+1}.$$ \hspace{1cm} (5)

KAMA shortens or extends the time period used for computing the moving average according to the conditions that prevail in the system. KAMA becomes more sensible or robust tightly connected with the frequency of the heartbeat messages received.

3) To make SC less sensible and to avoid a zigzag evolution, the final smoothing constant is obtained by squaring SC:

$$\alpha = SC^2.$$ \hspace{1cm} (6)

4) The final KAMA computation looks similar to EMA calculation by approximating the new predicted value for receiving a new heartbeat message based on the last received heartbeat value and the previous prediction:

$$P_i = \alpha \times A_{i-1} + (1 - \alpha) \times P_{i-1}.$$ \hspace{1cm} (7)

$$P_i = P_{i-1} + \alpha \times (A_{i-1} - P_{i-1}).$$

KAMA belongs to less known moving averages and its main advantage is that it takes into consideration not just the direction, but the heartbeat time volatility, as well. KAMA adjusts its length according to the prevailing keepalive signaling conditions and informs us about trends prevailing during parallel chat evaluations.

5) The final step in the evaluation of each worker’s current status is the determination of the suspicion level that ranges from 0 to 1. While a small value close to 0 represents a fully functional and working process, the closer the value gets to 1, the greater the probability of a malfunction. Therefore, with each missed or delayed heartbeat message, the level of doubt increases. In order to estimate the suspicion level we use the normalized value of the current time divided by predicted time:

$$s(t) = \frac{e^{\alpha(t-1)} - 1}{e^{\alpha(t-1)} + 1}.$$

Figure 3. Suspicion level evolution for different β values

Starting from the evolution of the suspicion level depicted in Figure 3, the imposed threshold is 0.8. By surpassing this value, the process is considered malfunctioning / unavailable and the task is reassigned. Fine refinement of the function can be obtained by adjusting the values of β. Therefore, for smaller values, more time representing a greater number of missed keepalive messages is allowed to pass until the master process declares the process dead. The value for β used in current tests was of 0.5, enabling a maximum of 5 heartbeat messages to be missed.
If a worker’s suspicion level exceeds the threshold, it is considered malfunctioning and its assigned task is reinitialized, later to be solved by a functional worker.

Because the only single point of failure in the described architecture is the master process, checkpoints are made after each completed task, enabling easy restore of the corpus assessment process in the eventuality the coordinator fails during its execution. Fast resume is possible in the case of an execution error in the master process by saving a snapshot each time a task is successfully completed.

B. Scenarios

Two scenarios were evaluated in order to assess the system’s performance. In the first case, only one worker was used, following a serial model execution. The results are briefly presented in Table I.

<table>
<thead>
<tr>
<th>No.</th>
<th>Worker</th>
<th>No. Tasks</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Machine1-52707-1278169861187-1:0</td>
<td>70</td>
<td>12.904,12</td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td></td>
<td>12.904,12</td>
</tr>
<tr>
<td></td>
<td>Execution time</td>
<td></td>
<td>12.904,12</td>
</tr>
</tbody>
</table>

The second scenario used the following assumptions: two identical machines were used, and at a given time, both workers had a simulated internal error and were terminated. Work was resumed on both machines and the corpus was fully evaluated. Table II depicts the results obtained in case of distributed execution of chat assessments.

<table>
<thead>
<tr>
<th>No.</th>
<th>Worker</th>
<th>No. Tasks</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Machine1-53513-1278188991704-1:0</td>
<td>28</td>
<td>5.136,66</td>
</tr>
<tr>
<td>2</td>
<td>Machine2-53515-1278188995545-1:0</td>
<td>24</td>
<td>5.327,69</td>
</tr>
<tr>
<td>3</td>
<td>Machine1-54936-127822494014-1:0</td>
<td>10</td>
<td>1.471,97</td>
</tr>
<tr>
<td>4</td>
<td>Machine2-54938-127822497153-1:0</td>
<td>8</td>
<td>1.559,32</td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td></td>
<td>13.495,64</td>
</tr>
<tr>
<td></td>
<td>Execution time</td>
<td></td>
<td>6.887,01</td>
</tr>
</tbody>
</table>

Execution time is expressed as the maximum value of total execution time for each machine.

C. Performance evaluation of distributed architecture

Although a slight decrease in overall time can be observed in the second case because of communication and internal processes, the increase in performance is quite remarkable: the concurrent execution time is of 6.887,00 seconds, speedup is 1,88 with a maximum value of 2 for two machines. Therefore the performance gain after using 2 parallel processes is of approximately 94%, with a deviation from “optimal” task assignment (total number of chats = 70 divided by the number of workers = 2) of 8,57% for each worker.

Dual-core machines were used in both scenarios allowing multiple threads to run concurrently without any issues regarding context switching. In the case of workers we have an execution thread and a keepalive sending thread, whereas in the case of a master process we have a monitoring thread for receiving heartbeat messages and a local task allocation and scheduling component. In both scenarios, a core was used almost at full load for computational purposes regarding the actual evaluation of the chat and about 1GB of RAM were also allocated.

Figure 4 highlights the actual CPU Usage on a worker station – one core is intensely used for processing chats whereas the other is responsible for keepalive messages and other system background processes.

Figure 5. Memory usage history

D. Final remarks regarding the distributed architecture

The main benefits of this architecture are robustness, adaptability to worker heartbeat messages and scalability allowing virtually an unlimited number of workers to be assigned on individual tasks.

The proposed architecture is effective because we are dealing with independent tasks, highly computational, with little I/O and little communications. Besides periodic keepalive messages, communication is present at the beginning, when each worker loads its configuration and receives an assigned task, and at the end of each task, when workers submit their
results. All messages - hello, start task, finish task and periodic heartbeat – are very small in dimension, making communication costs very small.

On the other hand, the speedup of the architecture is considerable and can be estimated to the actual number of workers multiplied by a factor of 94%. This value expresses latency with regards to communications and internal context switching due of other running threads.

IV. VALIDATION

Taking into consideration the average time of automatically assessing a chat (3 to 10 minutes) and the possibility of evaluating multiple discussions simultaneously, the designed system offers a great support for tutors in their learning context. The utility of the system was validated using a group of 5 tutors and 9 senior students that studied the Human-Computer Interaction course available at our university. The students were divided into two groups that had to debate which is the best web collaboration technology. After that, both students and the tutors used the system and provided a preliminary feedback about its usefulness in the evaluation process. All the 35 questions addressed to the tutors received average scores between 3.50-5.00 (where 5 is the maximum score and 1 the minimum), while the students validated 28 out of 32 questions with average scores between 3.66-5.00. These scores emphasize the utility of the presented instrument in the pedagogical process.

With regards to the evaluation process of a single conversation, we believe that performance and accuracy will improve by further tuning the weights of each factor, by providing better LSA learning and by integrating an increased number of social network factors.

V. CONCLUSION

The need for developing a tool for automatically assessing instant messaging conversations has become imperative due to the rapid increase in use and evolution of Computer Supported Collaborative Learning technologies. In this paper we have presented such an instrument, designed to evaluate chats using advanced techniques based on tagged latent semantic analysis, social network analysis and natural language specific processing. We presented results proving that the system is capable of evaluating a corpus of chats in a timely manner, granting fast access to feedback for participants. We also presented implementation details on a distributed version of the instrument, a solution that significantly increases the overall performance, allowing larger corporuses to be analyzed in a smaller amount of time. The system performs and scales well under a wide variety of conditions and loads.

Another benefit of the distributed architecture that is currently under implementation represents the possibility of building concurrently the term-document matrix used within the LSA learning stage.

ACKNOWLEDGMENT

The research presented in this paper was partially supported by the FP7 EU STREP project LTILL and the national CNCSIS grant K-Teams. The work has been co-funded by national project: ‘TRANSYS – Models and Techniques for Traffic Optimizing in Urban Environments’, Contract No. 4/28.07.2010, Project CNCSIS-PN-II-RU-PD ID: 238.

REFERENCES