論文の内容の要旨

論文題目　　ParaLite: a Parallel Database System for Data-intensive Workflows
(ParaLite: データ集約的ワークフローのための並列データベースシステム)

氏　　名　　陳　婷

Data-intensive workflows have become one of the most important and necessary tools for data-intensive applications since they facilitate the composition of individually developed executables, making it easier for domain experts to focus on their research rather than computation managements. A workflow generally consists of a set of jobs with their dependencies. Since a job is typically an existing executable, data transfers between jobs are generally handled by the workflow system. Usually, data are stored in files and implicitly transferred through a shared file system or explicitly moved by a staging subsystem. Such file-based workflows are often very complex with many jobs due to the low-level description. To schedule a job to computing resources for parallel execution, the input of the job is generally split into multiple small files, thus, leading to a large number of intermediate files.

While there is a critical need for workflow systems to manage scientific applications and data, parallel database systems which have been commercially available for decades and proved to be efficient large-scale data processing platforms, are well-suited to deal with specific aspects of workflow management. Some workflow management systems utilize database technologies to provide functionality such as simplifying the description of a workflow with SQL queries, improving the performance of the execution and facilitating the management of data. While database systems with high-level SQL queries simplify the description of workflows, they generally lack a good support for directly invoking executables from SQL statements. Many of executables are third-party components that received a large amount of development efforts from the community and usually developed in a variety of languages. As a workflow is typically built out of such executables, integrating them into SQL statements is very important. Most databases execute the executables in the form of user-defined functions or stored procedures. Thus, programmers who want to invoke such executables as part of SQL statements have to write and compile them with respect to the strict specifications of databases, and are usually constrained in the language they can use. It is obviously unreasonable for scientists to rewrite their applications with a large number of such executables to allow them to be run by a database. Another limitation for database systems for workflows is inefficient fault tolerance mechanisms. The conventional

approach in most existing database systems which handle failures by aborting the query and restarting it from the beginning, is not efficient for long-running jobs in workflows.

To tackle these problems, we propose ParaLite, a shared-nothing parallel database system which facilitates the development of workflows and improves the performance of their executions. The basic idea behind ParaLite is to provide a coordination layer to glue many SQLite instances together, and parallelize an SQL query across them. With ParaLite, jobs in a workflow are expressed with SQL queries and all intermediate data are stored as relational tables. To allow the direct invocation of external executable from SQL statements, ParaLite provides seamless integrations of external executables (User-Defined Executable, UDX for short) into SQL statements. The syntax of an UDX is similar to that of a User-Defined Function (UDF) but more flexible in the format of input and output data. With the support of UDX, programmers do not need to write any program with respect to strict specifications of databases. To provide efficient parallel execution of UDXes, ParaLite is equipped with a concept of collective query, an SQL query issued by multiple computing clients who collectively receive the results of the query and process them in parallel using UDXes. Collective query enables the co-allocation of computing clients and data sources (data nodes in databases) with consideration of data locality and load balance across all clients. With collective queries, the execution of an UDX is not bound to database nodes and it can be distributed to arbitrary clients for larger scale execution and computational load balancing.

Moreover, for long-running jobs in a workflow, ParaLite supports intra-query fault tolerance with a selective checkpointing mechanism, enabling to resume queries from middle of the execution upon a failure. Each query is represented by a DAG of relational operators in which data are typically pipelined between operators. The goal of the mechanism is to find a set of operators whose outputs are worth being checkpointed to minimize the expected completion time of the whole query. It firstly provides a cost model to estimate the expected completion time of a whole query plan under a given failure probability for each operator. Then a divide-and-conquer algorithm is proposed to find a close-to-optimal solution to the problem. The algorithm divides the query plan into sub-plans with smaller search spaces. For a given query plan with n operators, the algorithm runs in $O(n)$ time.

The experimental results firstly show that while ParaLite has similar performance with a commercial database system DBMS-X for most queries from TPC-H benchmark, it is 10x speedup comparing to UDF implementation in DBMS-X for the execution of executables. Besides, ParaLite has several times higher performance than a MapReduce system (specifically Hive) for typical SQL tasks, such as selections, joins and aggregations. With collective queries the performance for the UDX's execution could achieve close-to-ideal speedup with the increase of computing clients when data are either balanced or not balanced distributed across a cluster. Moreover, the mechanism of collective query balances the load across computing clients even when some clients are manually overloaded. The experimental results also indicate that different fault-tolerant strategies affect the overall runtimes of queries. Our selective checkpointing mechanism can choose reasonable operators to be checkpointed and outperforms other fault-tolerant strategies, such as pure pipelining data and checkpointing all intermediate data. In

addition, the divide-and-conquer algorithm taken by our mechanism has a smaller overhead than brute-force approach while keeping a similar effectiveness.

Finally, we study three real-world text-processing workflows in the field of Natural Language Processing (NLP), and build them on top of ParaLite, Hadoop, Hive and regular files. We discuss their strengths/weaknesses both in terms of programmability and performance for each workflow. Our development experience reveals that high-level query languages such as SQL of ParaLite and HiveQL of Hive are helpful for expressing data selection, join, aggregation and calculation by typical executables. In NLP workflows, the expressiveness of SQL in ParaLite is particularly useful since it provides natural supports of file-based NLP executables and reusing existing NLP tools by tracking the association between a document and its annotation attached by the tools. On the other hand, workflows expressed in low-level languages lack good support of all features mentioned above, requiring a few extra efforts. The experimental results show that essentially each system has a similar overall performance because performing executables takes most of time. However, a closer investigation still reveals a potential advantage of ParaLite due to data partitioning and query optimization.