

An Oracle White Paper
October 2009

In-Database Map-Reduce

Introduction	2
The Theory	3
Step-by-Step Example	4
Step 1 – Setting up the Environment	4
Step 2 – Creating the Mapper	6
Step 3 – A Simple Way of Using the Mapper	7
Step 4 – Writing the Reducer	7
Step 5 – In-Database Map-Reduce.....	9
Summary	10
Appendix 1 – Package Header	11
Appendix 2 – Package Body	12

Introduction

The Map-Reduce model has become a popular way for programmers to describe and implement parallel programs. These custom map-reduce programs are often used to process a large data set in parallel. This paper shows how to implement Map-Reduce Programs within the Oracle database using Parallel Pipelined Table Functions and parallel operations.

The Theory

Pipelined Table Functions were introduced in Oracle 9i as a way of embedding procedural logic within a data flow. At a logical level, a Table Function is a function that can appear in the FROM clause and thus functions as a table returning a stream of rows. Table Functions can also take a stream of rows as an input. Since Pipelined Table Functions are embedded in the data flow they allow data to be 'streamed' to a SQL statement avoiding intermediate materialization in most cases. Additionally, Pipelined Table Functions can be parallelized.

To parallelize a Table Function the programmer specifies a key to repartition the input data. Table Functions can be implemented natively in PL/SQL, Java, and C. You can find more information and examples about Table Functions and the functionality mentioned above at the following URL:

http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96624/08_subs.htm#19677

Pipelined Table Functions have been used by customers for several releases and are a core part of Oracle's extensibility infrastructure. Both external users and Oracle Development have used Table Functions as an efficient and easy way of extending the database kernel.

Examples of table functions being used within Oracle are the implementation of a number of features in Oracle Spatial and Oracle Warehouse Builder. Oracle Spatial usages include spatial joins and several spatial data mining operations. Oracle Warehouse Builder allows end users to leverage Table Functions to parallelize procedural logic in data flows such as the Match-Merge algorithm and other row-by-row processing algorithms.

Step-by-Step Example

To illustrate the usage of parallelism, and Pipelined Table Functions to write a Map-Reduce algorithm inside the Oracle database, we describe how to implement the canonical map-reduce example: a word count. For those unfamiliar with the example, the goal of word count is to return all distinct words within a set of documents as well as a count of how often this word occurs within this set of documents.

The procedural code in this word count example is implemented in PL/SQL but, as said before, Oracle allows you to pick your language of choice to implement said procedural logic.

Step 1 – Setting up the Environment

We will be looking at a set of documents, these documents can be either files outside of the database, or they can be stored as Secure Files/CLOB columns within the database. Within this table our documents are stored, effectively reflecting a file system.

In this case we are going to create an table within the database using the following definition:

```
CREATE TABLE documents
(a CLOB)
LOB(a) STORE AS SECUREFILE(TABLESPACE sysaux);
```

Each row in this table corresponds to a single document. We populate this table with a very simple corpus resulting in 3 documents with the text shown here:

```
INSERT INTO documents VALUES ('abc def');
INSERT INTO documents VALUES ('def ghi');
INSERT INTO documents VALUES ('ghi jkl');
commit;
```

The end result of both the map function and the reduce table function are going to live in a package, keeping the code nice and tidy. To show the steps to be taken we will take snippets from the overall package and show those in the section to follow. The actual package will contain a set of types, which are required for the code to work. All code was tested on Oracle Database 11g (11.1.0.6). The following figures show the package being deployed.

```
C:\WINDOWS\system32\cmd.exe - sqlplus oe/oe
SQL> create or replace
2 package oracle_map_reduce is
3
4     type word_t      is record (word varchar2(4000));
5     type words_t     is table of word_t;
6
7     type word_cur_t  is ref cursor return word_t;
8     type wordcnt_t   is record (word varchar2(4000), count number);
9     type wordcnts_t  is table of wordcnt_t;
10
11     function mapper(doc in sys_refcursor, sep in varchar2) return words_t
12         pipelined parallel_enable (partition doc by any);
13
14     function reducer(in_cur in word_cur_t) return wordcnts_t
15         pipelined parallel_enable (partition in_cur by hash(word))
16         cluster in_cur by (word);
17
18 end;
19 /
Package created.
SQL>
```

Figure 1. Creating the package header

```
C:\WINDOWS\system32\cmd.exe - sqlplus oe/oe
74         pipe row (word_count);
75         word_count.word := next;
76         word_count.count := 1;
77
78     else
79
80         word_count.count := word_count.count + 1;
81
82     end if;
83
84 end loop;
85
86 if word_count.count <> 0 then
87     pipe row (word_count);
88 end if;
89
90 return;
91
92 end reducer;
93
94 end;
95 /
Package body created.
SQL> _
```

Figure 2. Creating the package body

Step 2 – Creating the Mapper

First we need to create a generic function to “map” (as in map-reduce) or tokenize a document. Note that the goal is not to show the best map function, but how this will work in principle in the database. This specific map function is very basic and better implementations may be found elsewhere. If you are doing this on a different table, substitute your name into all of the following code.

```
function mapper(doc in sys_refcursor, sep in varchar2) return
words_t
  pipelined parallel_enable (partition doc by any)
  is
    document clob;
    istart   number;
    pos      number;
    len      number;
    word_rec word_t;
  begin

    -- for every document
    loop

      fetch doc into document;
      exit when doc%notfound;

      istart := 1;
      len := length(document);

      -- For every word within a document
      while (istart <= len) loop
        pos := instr(document, sep, istart);

        if (pos = 0) then
          word_rec.word := substr(document, istart);
          pipe row (word_rec);
          istart := len + 1;
        else
          word_rec.word := substr(document, istart,
                                  pos - istart);
          pipe row (word_rec);
          istart := pos + 1;
        end if;

      end loop; -- end loop for a single document

    end loop; -- end loop for all documents

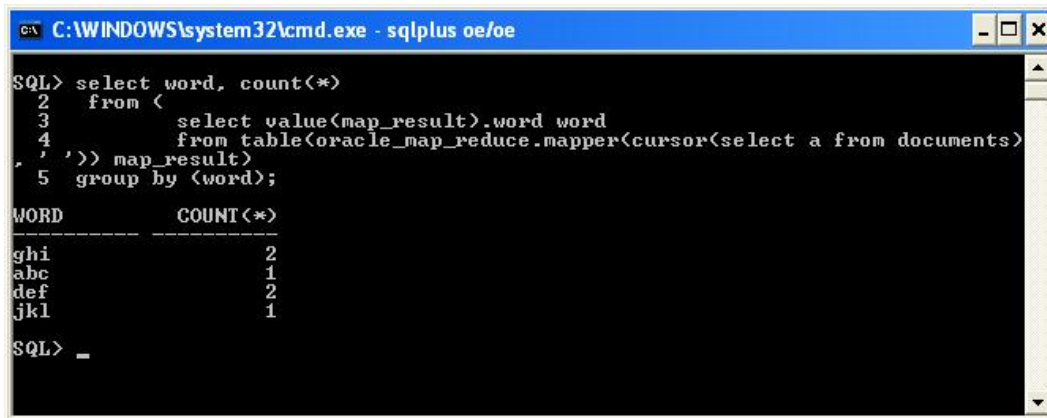
    return;
  end mapper;
```

Step 3 – A Simple Way of Using the Mapper

Now that we have the mapper available in the database we can make use of it (note, create the package in Appendix 1 and 2 before running the select statement). Running the following statement would give us a list of all the words in all the documents in the **documents** table and a count of the number of occurrences for each. This specific statement uses the Oracle's aggregation engine to do the counting. In map-reduce you will not use the Oracle engine but write your own aggregation (the reducer).

```
select word, count(*)
  from (
        select value(map_result).word word
        from table(oracle_map_reduce.mapper(cursor
          (select a from documents), ' ')) map_result)
 group by (word);
```

The resulting output is shown here:



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe - sqlplus oe/oe". The prompt is "SQL>". The user enters the following SQL query:

```
SQL> select word, count(*)
2   from (
3       select value(map_result).word word
4       from table(oracle_map_reduce.mapper(cursor(select a from documents)
5       , ' ')) map_result)
6   group by (word);
```

The output is a table with two columns: "WORD" and "COUNT(*)". The data rows are:

WORD	COUNT(*)
ghi	2
abc	1
def	2
jkl	1

The prompt returns to "SQL> _".

Figure 3. List of words and their counts using the map function

Step 4 – Writing the Reducer

Of course, you could, write your own aggregation Table Function to count the occurrences of words in a document. That is what you would do if you were writing the map-reduce program without leveraging the Oracle aggregation engine as we did before. This aggregation Table Function is the reducer of the map-reduce program unit.

The Table Function specifies that it's input be partitioned by word and could (to use the Oracle execution engine's sort) ask for the data to ordered or clustered by word. We show a sample count program here to complete the example.


```
function reducer(in_cur in word_cur_t) return wordcnts_t
  pipelined parallel_enable
    (partition in_cur by hash(word))
  cluster in_cur by (word)
is
  word_count wordcnt_t;
  next        varchar2(4000);
begin

  word_count.count := 0;

  loop

    fetch in_cur into next;
    exit when in_cur%notfound;

    if (word_count.word is null) then

      word_count.word := next;
      word_count.count := word_count.count + 1;

    elsif (next <> word_count.word) then

      pipe row (word_count);
      word_count.word := next;
      word_count.count := 1;

    else

      word_count.count := word_count.count + 1;

    end if;

  end loop;

  if word_count.count <> 0 then
    pipe row (word_count);
  end if;

  return;

end reducer;
```

Step 5 – In-Database Map-Reduce

With this we have created our complete map-reduce program. Running a query using this Table Function will give us a parallel workload on external documents, doing what the typical map-reduce programs do.

```
select * from table(oracle_map_reduce.reducer(
    cursor(select value(map_result).word word
    from table(oracle_map_reduce.mapper(
        cursor(select a from documents), ' '))
    map_result)));
```

```
SQL> select *
2   from table(oracle_map_reduce.reducer(
3       cursor(select value(map_result).word word
4       from table(oracle_map_reduce.mapper(
5           cursor(select a from documents), ' '))
6       map_result)));
```

WORD	COUNT
abc	1
def	2
ghi	2
jkl	1

```
SQL> _
```

Figure 4. Results from a custom Table Function

As these set of simple steps show, building a map-reduce program within the Oracle database is straightforward. By using Table Functions and a Table Function's ability to process all sorts of logic in parallel gives you an easy to use interface into your data and into parallel processing.

Summary

Oracle Table Functions are a proven technology, used by many internal and external parties to extend Oracle Database 11g.

Oracle Table Functions are a robust scalable way to implement Map-Reduce within the Oracle database and leverage the scalability of the Oracle Parallel Execution framework. Using this in combination with SQL provides an efficient and simple mechanism for database developers to develop Map-Reduce functionality within the environment they understand and with the languages they know.

Appendix 1 – Package Header

The following is the package header as shown before:

```
create or replace
package oracle_map_reduce is

    type word_t      is record (word varchar2(4000));
    type words_t     is table of word_t;

    type word_cur_t  is ref cursor return word_t;
    type wordcnt_t   is record (word varchar2(4000), count
number);
    type wordcnts_t  is table of wordcnt_t;

    function mapper(doc in sys_refcursor, sep in varchar2) return
words_t
        pipelined parallel_enable (partition doc by any);

    function reducer(in_cur in word_cur_t) return wordcnts_t
        pipelined parallel_enable (partition in_cur by hash(word))
        cluster in_cur by (word);

end;
/
```

Appendix 2 – Package Body

The following code shows the package body as used in this example (continued over two pages):

```
create or replace
package body oracle_map_reduce is

  --
  -- The mapper is a simple tokenizer that tokenizes the input
  documents
  -- and emits individual words
  --
  function mapper(doc in sys_refcursor, sep in varchar2) return
  words_t
  pipelined parallel_enable (partition doc by any)
  is
    document clob;
    istart  number;
    pos     number;
    len     number;
    word_rec word_t;
  begin

    -- for every document
    loop

      fetch doc into document;
      exit when doc%notfound;

      istart := 1;
      len := length(document);

      -- For every word within a document
      while (istart <= len) loop
        pos := instr(document, sep, istart);

        if (pos = 0) then
          word_rec.word := substr(document, istart);
          pipe row (word_rec);
          istart := len + 1;
        else
          word_rec.word := substr(document, istart, pos - istart);
          pipe row (word_rec);
          istart := pos + 1;
        end if;

      end loop; -- end loop for a single document

    end loop; -- end loop for all documents

    return;

  end mapper;

  -- code continues...
```

```
-- The reducer emits words and the number of times they're seen
--
function reducer(in_cur in word_cur_t) return wordcnts_t
  pipelined parallel_enable (partition in_cur by hash(word))
  cluster in_cur by (word)
is
  word_count wordcnt_t;
  next        varchar2(4000);
begin

  word_count.count := 0;

  loop

    fetch in_cur into next;
    exit when in_cur%notfound;

    if (word_count.word is null) then

      word_count.word := next;
      word_count.count := word_count.count + 1;

    elsif (next <> word_count.word) then

      pipe row (word_count);
      word_count.word := next;
      word_count.count := 1;

    else

      word_count.count := word_count.count + 1;

    end if;

  end loop;

  if word_count.count <> 0 then
    pipe row (word_count);
  end if;

  return;

end reducer;

end;
/
```



White Paper TitleIn-database Map-Reduce
October 2009

Authors: Shrikanth Shankar, Jean-Pierre Djicks

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com



| Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.