DS/390 Benchmark Report
***** DRAFT *****

Summary

Ascential Software and IBM participated in a benchmark of DataStage DS/390 Version 6 in the IBM Montpellier Benchmark Center during August and September, 2002. The purpose of the benchmark was to demonstrate that DS/390 can generate highly-performant COBOL code, enabling customers to build total end-to-end ETL solutions involving S/390 data without requiring movement of the data to distributed systems. Our strategy was to measure the ability of DataStage generated COBOL programs to move and translate large data volumes (up to 1 Terabyte), demonstrate the ability to run the COBOL programs in parallel, demonstrate the ability to incorporate the OS/390 Batch Pipe facility into the process flows, and to provide some data points that could be used by IBM and Ascential customers to begin to estimate DataStage DS/390 resource requirements (equipment sizing). This benchmark was not intended to compare the OS/390 and UNIX versions of DataStage, the IBM z-Series and p-Series hardware, or the UNIX and OS/390 versions of DB2.

The jobs used to perform the benchmark tests are from Ascential's Enterprise Data Integration Benchmark and are documented on the Ascential web site **(http://www.ascentialsoftware.com/etlbenchmark)**. The Ascential benchmark suite is more representative of real applications than competitive benchmarks, including reference lookups, date reformatting, and a variety of data transformations such as character string manipulations, mathematical operations, and data type conversion.

The results show that DataStage generated COBOL programs without transformations can move data at roughly the same speed as the IBM IEBGENER utility. Further, the results demonstrate that I/O considerations, such as BLOCKSIZE, BUFNO, and I/O bandwidth can have dramatic effects on ETL throughput. The DataStage COBOL programs exhibited linear scalability with respect to data volumes. Use of the MVS Batch Pipes facility can have a significant positive impact on throughput. In fact, a single DS/390 COBOL program, using the Batch Pipe facility, loaded the entire 1 TB of data into DB/2 in less than 18 hours and 30 minutes. Finally, both DB/2 and DS/390 show very similar performance attributes when repartitioning data.

These results support the assertion that customers with data existing on the OS/390 platform can use DS/390 generated COBOL programs to address their ETL needs without moving the data off of the OS/390 environment. DataStage offers customers (whether mainframe or distributed) a solution that will satisfy their requirements, and perform extremely well. This is not to state that DS/390 is the only solution, but that DS/390 provides customers with the option of manipulating the data in the environment that makes most sense. Large OS/390 data volumes, OS/390 targets, or the ability to integrate or filter (reduce) the data on the OS/390 platform before moving it are all reasons to consider DS/390.

Configuration Details

The benchmark was performed on an IBM 2064 model 216 zSeries machine, with 16 processors providing an aggregated speed of over 3400 MIPS. Our machine was configured with 32GB of main memory and twelve 2 Gbit FICON channels to attach to the I/O subsystems. A FICON channel is theoretically equivalent to 3 or 4 ESCON channels and provides a 100 MB/sec bidirectional (full duplex) data transfer rate. For our tests we attached 3 FICON channels to each cluster on two 3+ Terabyte IBM Enterprise Storage Subsystems (Sharks) that were used to contain the input and output data respectively. This meant the each Shark had 6 FICON channels attached to it.

The software used for the benchmark consisted of z/OS 1.3, DB/2 V7.1, and DataStage/XE 390 V6. DB/2 was configured with 32K pages and 16, 32, or 64 partitions, depending on the tests being run. DB/2 logging was enabled but logs were not archived.

The ESS device (Shark) can be configured to offer a wide range of logical device architectures, for our testing we decided to use 8 logical control units per Shark and 3390 model 9 format disks (7.8 GB/volume) to reduce the number of logical volumes necessary to manage (one 3390 model 9 is equivalent to three 3390 model 3's). Configured at 2.7GB/disk (Model 3 format) we would have needed about 384 disks to accommodate the terabyte, using mod 9s only required 128 disks - a much more manageable number.

The Shark offers a number of other optional features– the only feature we decided to exercise during the benchmark was the Parallel Access Volume (PAV) functionality. PAV allows z/OS to queue multiple I/O requests for the same logical volume, reducing job contention and improving the overall system performance for I/O constrained workloads.

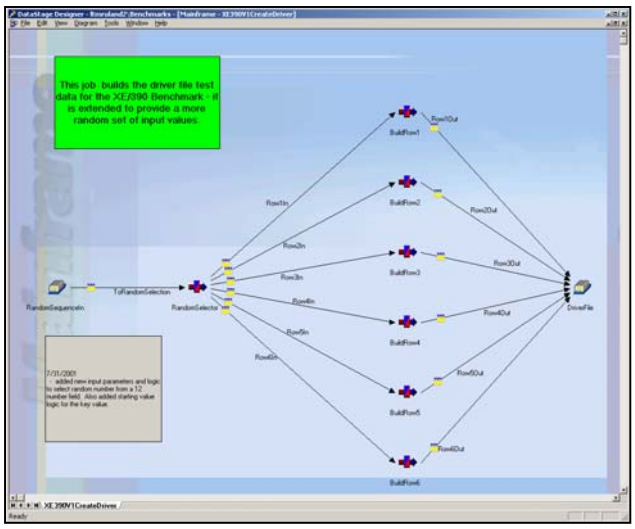
We also took advantage of the z/OS System Managed Storage (SMS) facility to control the layout of the input data and intermediate load ready files. This facility dramatically reduces the need to manually place the data for performance as is typically done in UNIX benchmarks and more closely represents the way our customers would run in their environments.

Data Generation

The first task was generating the data and DataStage was used for this phase of the benchmark as well. The DS/390 COBOL programs were used to generate 128 datasets of 8GB each. This configuration was chosen to allow us to run both low volume tests (8, 16, and 32 GB), as well as high volume tests (512, 768, and 1024 GB [1 TB]).

The generation process performed very well with the DS/390 COBOL jobs creating a Terabyte of input data in less than 1 hour 20 minutes. We ran the jobs in two waves of

64 simultaneous jobs each. DS/390 itself does not provide a parallelism capability such as offered by DS PX. The jobs were run in parallel by parameterizing the input values and using a REXX exec to point to the appropriate job parameter file and submit the job sets. This does not match exactly the capability of the PX, but it allowed us to apply similar concepts to the OS/390 environment. The data generation jobs stressed the throughput of the Shark to over 240 MB/second. This same technique was used to run many of the remaining benchmark tests.

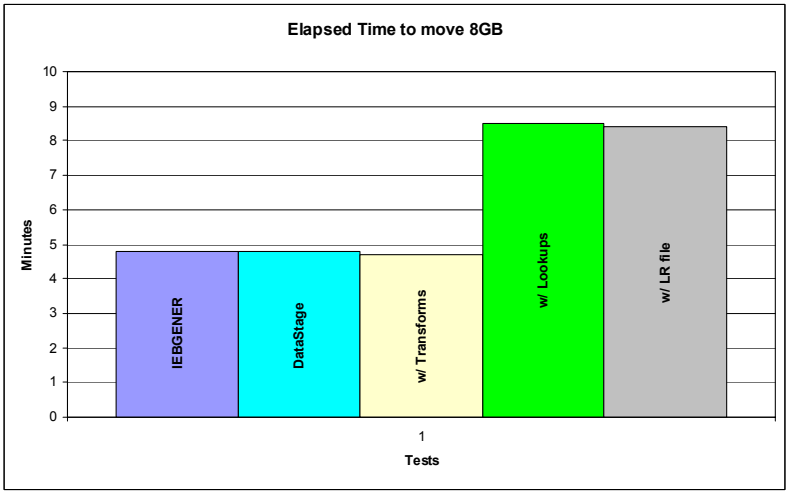


The diagram to the left shows the DS/390 job flow that generated the test data. The goal of the generator is to create a random set of input records containing a unique primary key and a lookup key that will yield an 80% hit ratio during the lookup operation. Each record contains 97 columns and consists of character, integer, and packed decimal fields.

Unit Testing

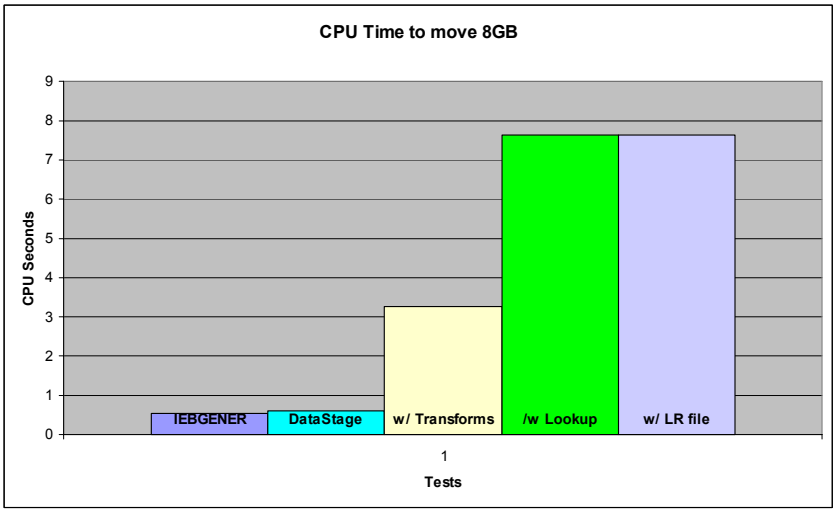
The benchmark environment was validated by running a number of simple data movement jobs using the IBM utility, IEBGENER. The IEBGENER run was then compared to various combinations of DataStage jobs to determine how data throughput was affected by various common data movement functions. The following tables show the results of these tests.

The first chart (below) shows the elapsed time for moving 8 Gigabytes of data between flat files. As you move from left to right on the chart, additional functionality is continually added to the process. For example, the last test (w/ LR file) reads the flat file, performs the all required transformations and lookups, then finally creates a DB2 load ready file. As the chart shows,

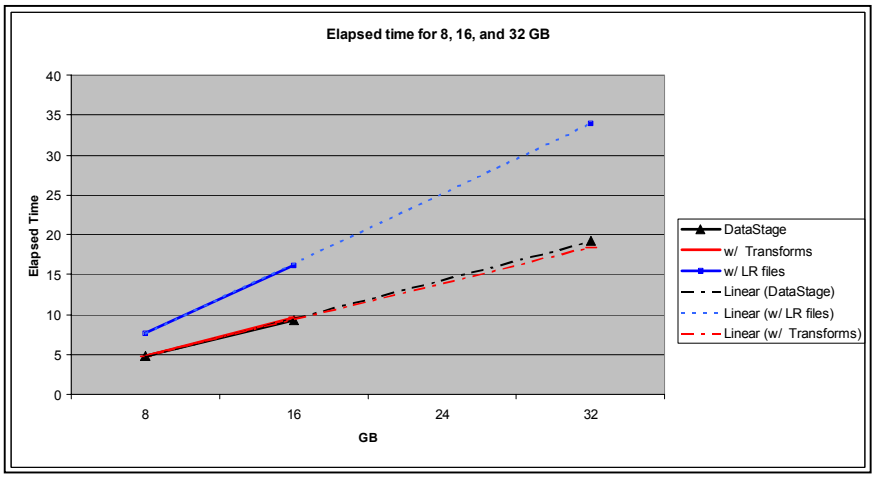


DataStage performed the movement operation in roughly the same amount of time required by the OS/390 IEBGENER utility. Also of note is that creating the DB/2 Load Ready file (a comma separated format file) did not add significant resource requirements to the job.

The next chart shows the same tests as before this time reporting on total CPU seconds consumed (TCB+SRB). Here we see a different view of the situation, more telling than simply looking at the rows/sec metric. Transformations are indeed adding a certain amount of CPU time to the operation, demonstrating that the transformations and lookups are not trivial – some real work is being done here. The probable reason that the elapsed times shown in the previous chart are the same for the DS/390 straight move and the test with the transformations is that the additional CPU time is overshadowed by the I/O time. The in-memory lookup operation adds another increase in CPU usage. As we increase the number of COBOL programs running concurrently we should expect to see the CPU become constrained if the I/O subsystem can support the aggregated I/O load.

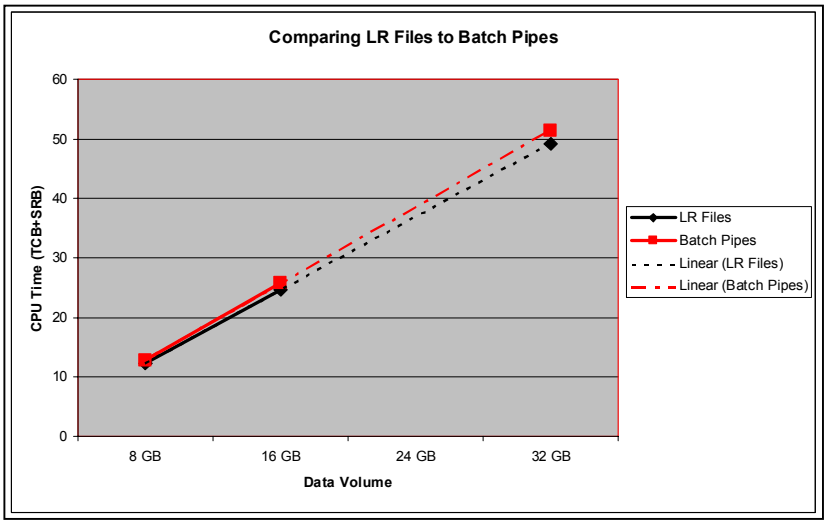


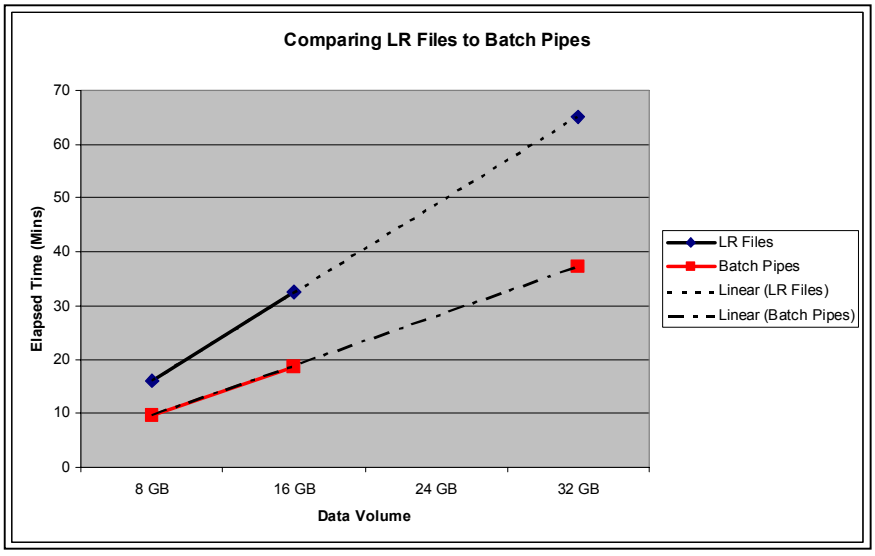
This diagram shows the results of running the DS/390 jobs at different data volumes (8, 16, and 32 GB). This chart begins to show the linear scalability of the COBOL programs generated by DS/390. Note how the slope of the "w/ LR" line is much different than that of the straight move test or the test involving the transformations. This is due to the extra work done for each row in performing the lookup operation. Additional logic was also added in this job to populate fields if they were not found during the lookup. This was done using IF/THEN/ELSE logic.



The next tests done focused on the usage of the MVS Batch Pipe facility instead of physically landing the DB/2 Load Ready file. The MVS Batch Pipe facility allows an OS/390 job to pass data to a second job through a shared memory buffer rather than physically landing the file. This capability is generally referred to as pipeline parallelism. While our results show that this facility can significantly reduce elapsed time, the trade off is the elimination of the flat file for use as a process restart point. The Batch Pipe facility also requires the DS/390 COBOL program and the DB/2 bulk load utility to both be running simultaneously, using an intermediate load ready file also allows operations to split the workload and run the DS/390 COBOL program independently from the DB/2 bulk load utility – as long as the COBOL program completes prior to the DB/2 load.

Our first chart illustrates the total CPU (TCB+SRB) consumption of both the COBOL program as well as the DB/2 bulk load utility for data volumes of 8, 16, and 32 GB of data. While Batch Pipes does use slightly more CPU seconds, the amount is negligible when compared to the next chart.





Compare the CPU seconds used chart above with the throughput chart shown to the left. Clearly the MVS Batch Pipe facility can be used to significantly reduce elapsed time of the ETL process. A further benefit here is that for large volumes, we have also eliminated the disk space necessary to hold the load ready file.

To demonstrate DS/390's ability to optimize lookups we performed a small test to illustrate the benefits of DS/390's lookup techniques. We used 99,999 records in a fixed width flat file (FWFF) against a lookup table consisting of the same 99,999 records. DS/390 has the ability to load the lookup file into memory and perform all lookups against the memory copy of the data – this facility is called a Hashed lookup since the lookup data is loaded into memory using a hashing technique. The tests were run 3 ways, using in-memory hashing for both a flat file and a relational table and using a direct read to the database against an indexed column. The results are presented in the next table.

Test Description	Elapsed time
Using FWFF for the reference file, Hash lookup	.19 min
Using DB2 for the reference file, Hash lookup	.30 min
Using Auto lookup against an indexed DB2 table	.46 min

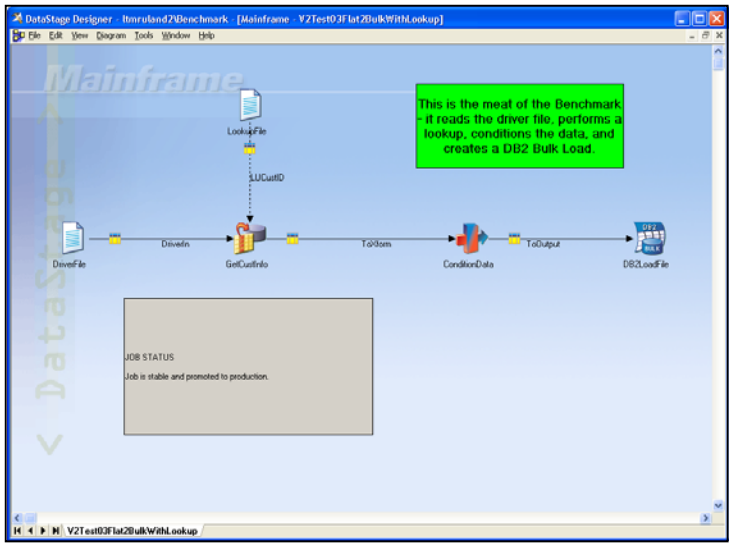
This clearly illustrates that the most efficient lookup technique is using FWFF for the reference files and the in-memory hashing technique. Next most efficient is using DB2 as the reference, but still hashing. The least efficient is Auto lookup using the DB2 index.

A short test was also performed to demonstrate the dramatic effect that blocksize and buffer numbers can have on run times. These tests performed all the transformations and lookups, creating a DB/2 load ready file. The tests were run on a small test sample input size (30,000 rows), but with the standard lookup files.

Test Description	Elapsed Time
Out of the box - no changes made	0:01:30
Increased BUFNO on the Driver and LR file for the DataStage step to 100	0:00:44
Increase LR blocksize to 32273, decreased bufno to 50 and added bufno 50 to sysrec in load step	0:00:47
Decreased LR file blocksize to 22974 (1/2 track blocking) to reduce file size. Removed bufno=50 from the sysrec dd statement.	0:00:47
Added bufno=100 to lookup file and removed bufno from the load ready file	0:01:08
readded bufno=50 to the load ready file	0:00:47
changed blocksize to 3829 and 100 buffers on the load ready file	0:00:55

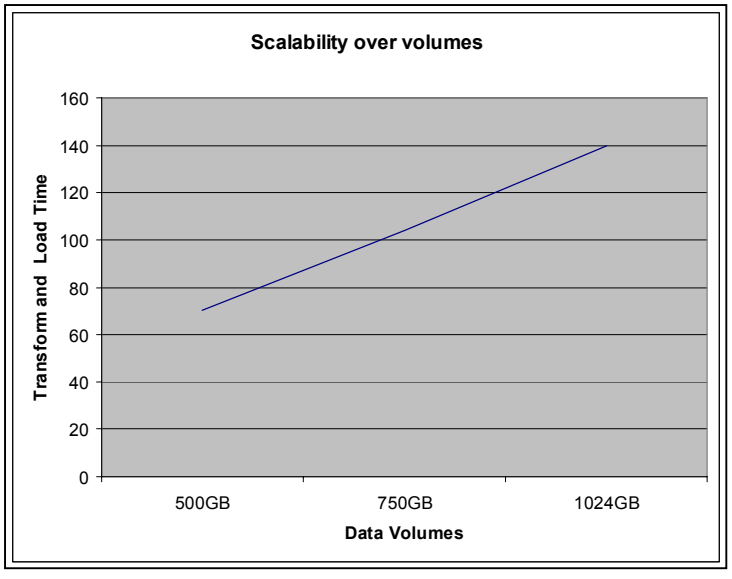
This test sequence clearly shows the dramatic effect that blocksize and buffering can have on throughput. Keep in mind that the test input sizes used here were tiny compared to the other runs we did. We also saw this dramatic effect when we ran the data movement job to compare to IEBGENER described earlier. Without specifying the blocksize the job ran in 6.8 minutes, compared to 4.8 minutes with the blocksize – a 42% increase in elapsed time. EXCP's also increased from 618K to 2560K – a 314% increase! CPU time (TCB+SRB) also increased 25%.

The Enterprise Data Integration Benchmark

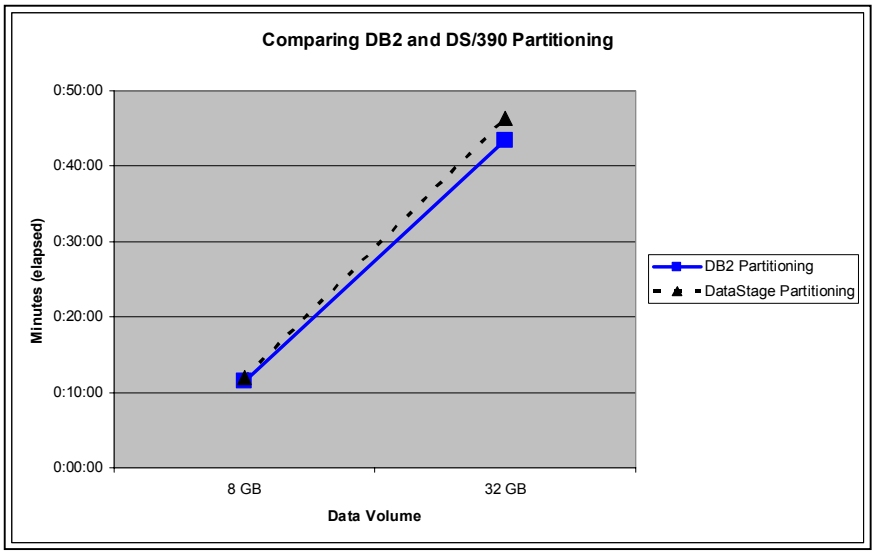


The diagram to the right illustrates the DS/390 process that implements the Ascential Enterprise Data Integration Benchmark. The graphic design capabilities of DS/390 make it easy to follow the flow of the data through the process. We can easily locate the input and lookup files as well as the process that implements the lookup conditions. The transformations are localized to the "ConditionData" stage, so that any changes or questions about the process can be easily answered by drilling into the metadata associated with that "stage". This is the job that was used for the volume tests, as well as that used in the "w/ LR file" tests discussed earlier.

The next diagram shows the results of running the benchmark job at high data volumes. This test was run reading from the flat file, performing a lookup with an 80% hit ratio, transforming approximately 68% of the fields, and then formatting the data into DB/2 load ready format. The data was not landed in this test, but instead passed the to MVS Batch Pipe facility, which presented it to the DB/2 bulk load utility. These tests were conducted running 32 simultaneous COBOL programs. We did perform one further benchmark using the same process with a single COBOL program that loaded the full terabyte of data into DB/2 in 18 hours and 21 minutes.

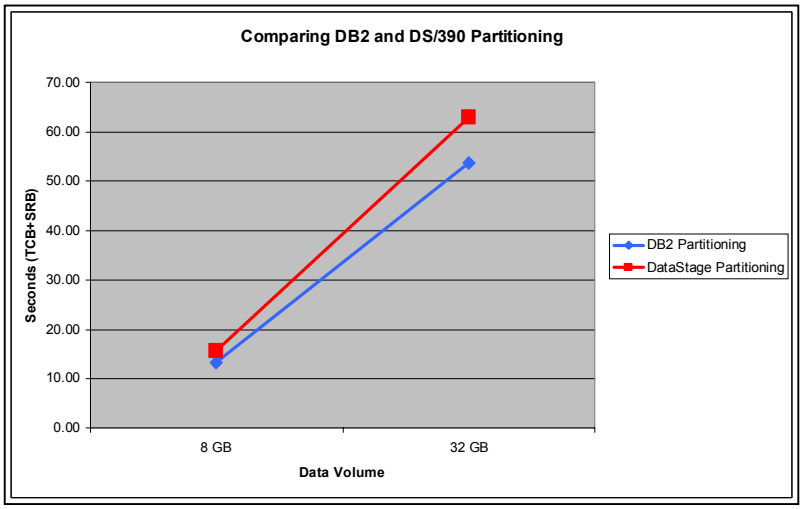


One of the fundamental requirements for running parallel environments is the ability to repartition the data as it flows through the process. To shed some light on this aspect of running large volume data transformations in the OS/390 environment, we set up a test to compare DB/2 data partitioning with DataStage partitioning. For this test DB/2 was partitioned 16 ways and the input data was regenerated to randomize the order in which it was read. In the DB/2 partitioning test, a single DS/390 COBOL program read the randomized data and passed it to a single instance of the DB/2 bulk loader for repartitioning. The DataStage repartitioning test was set up by changing the DataStage job to read the randomized input data and separate the data stream into16 different Batch Pipes. 16 instances of the DB/2 bulk loader were used to accept the data from the batch pipes. This test also validated the Batch Pipe facility's stability under moderately heavy I/O loads.



The first chart shows elapsed time for data volumes of 8, 16, and 32 GB of data. The elapsed time for the different partitioning schemes is reasonably close across these data volumes and the line indicates the process is linearly scalable.

Looking at total CPU consumption the lines also show linear scalability. The additional CPU consumption necessary to support 15 additional Batch Pipes (see our earlier discussion) and the 15 additional DB/2 bulk load processes begins to become noticeable as our volumes increase. In retrospect, it may have been interesting to perform this test using load ready files as well.



One final observation was with respect to dataset placement. We chose to initially configure the 2 Shark I/O subsystem by creating 3 main SMS pools. Pool 1 was defined on Shark1 and contained the input datasets, Pool 2 was defined on Shark 2 and contained the load ready files, and finally Pool 3 was defined on Shark1 again to contain the DB/2 table space. The reason for this configuration was we felt we would be reading from Shark1 and writing to Shark2 while the load ready files were being created; the DB/2 load would then reverse this and read from Shark2 while writing to Shark1.

We later decided to spread the data across both Sharks to determine if distributing the I/O load would improve throughput. The input files were placed so that a maximum of 2 jobs would be simultaneously accessing each logical control unit in each Shark. The DB/2 dataspaces were spread across both Sharks by SMS and the index spaces were placed on Shark 2. The runtimes in the following table show the result of this test. There are two tests represented in the table. The first 2 rows tested the effect of using the 2 different configurations for the input datasets. The second 2 rows in the table show the effect of the 2 different configurations for the output datasets.

Job	Elapsed Time
Optimized input disk	2:20:38
SMS input disk	2:20:01
LR files old output layout	2:59:33
LR files new output layout	2:45:58

This effect could be skewed somewhat by the high CPU busy times occurring during these runs and the fact that both Sharks were saturated. However, we can still draw the conclusion that SMS placement along with the PAV facility of the Sharks discussed earlier should do a reasonable job of managing data placement. The massive data volumes manipulated during this benchmark should not occur very frequently in most customer environments.

Conclusions

The purpose of the benchmark was to demonstrate that DS/390 can generate highly-performant COBOL code (the COBOL jobs used throughout the benchmark were run without modification), enabling customers to build total end-to-end ETL solutions involving S/390 data without requiring movement of the data to distributed systems. We believe the results shown in this document support this assertion. Our customers should feel confident that they can experience the productivity benefits of the latest generation of ETL technologies together with the stability and throughput of the z/OS and OS/390 environments.

Decision support is an enterprise wide effort and most enterprises continue to have a significant investment in data stored in the z/OS and OS/390 environments. DS/390 provides the ability to collect, cleanse/manipulate, and load the data in these environments, allowing you to perform all or some portion of these operations close to the data source when required. Eliminating the need to move the data source across a network connection can significantly simplify and lower the cost of building decision support structures.

Appendix A – JCL to create 32 partition DB/2 table space

```
//DSTAGEL JOB ,'DATASTAGE',CLASS=A,MSGCLASS=X,
//     REGION=0M,LINES=50000,NOTIFY=&SYSUID
//*********************************************************************
//****   DDL FOR CREATE TABLESPACES
//*********************************************************************
//*
//JOBLIB  DD  DSN=DSN710.SDSNLOAD,DISP=SHR
//        DD  DSN=DSN710.DBL0.SDSNEXIT,DISP=SHR
//*
//*
//PH01S01 EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  DSN SYSTEM(DBL1)
  RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP71) -
       LIB('DSNDBL0.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
 SET CURRENT SQLID='DSTAGE' ;
  DROP DATABASE DBDSTAGE ;
 COMMIT;
  CREATE DATABASE DBDSTAGE
    BUFFERPOOL BP32K2
    INDEXBP    BP3
    STOGROUP DBDST1;
  COMMIT;
  CREATE TABLESPACE TSBENCH1
    IN DBDSTAGE
    USING STOGROUP DBDST1
    PRIQTY 7200000 SECQTY 3600000
    FREEPAGE 0 PCTFREE 0
    DSSIZE 64 G
    NUMPARTS 32
     (PART 1 USING STOGROUP DBDST1 ,
      PART 2 USING STOGROUP DBDST1 ,
      PART 3 USING STOGROUP DBDST1 ,
      PART 4 USING STOGROUP DBDST1 ,
      PART 5 USING STOGROUP DBDST1 ,
      PART 6 USING STOGROUP DBDST1 ,
      PART 7 USING STOGROUP DBDST1 ,
      PART 8 USING STOGROUP DBDST1 ,
      PART 9 USING STOGROUP DBDST1 ,
      PART 10 USING STOGROUP DBDST1 ,
      PART 11 USING STOGROUP DBDST1 ,
      PART 12 USING STOGROUP DBDST1 ,
      PART 13 USING STOGROUP DBDST1 ,
      PART 14 USING STOGROUP DBDST1 ,
      PART 15 USING STOGROUP DBDST1 ,
      PART 16 USING STOGROUP DBDST1 ,
      PART 17 USING STOGROUP DBDST1 ,
      PART 18 USING STOGROUP DBDST1 ,
      PART 19 USING STOGROUP DBDST1 ,
      PART 20 USING STOGROUP DBDST1 ,
      PART 21 USING STOGROUP DBDST1 ,
      PART 22 USING STOGROUP DBDST1 ,
      PART 23 USING STOGROUP DBDST1 ,
      PART 24 USING STOGROUP DBDST1 ,
      PART 25 USING STOGROUP DBDST1 ,
      PART 26 USING STOGROUP DBDST1 ,
      PART 27 USING STOGROUP DBDST1 ,
      PART 28 USING STOGROUP DBDST1 ,
      PART 29 USING STOGROUP DBDST1 ,
      PART 30 USING STOGROUP DBDST1 ,
      PART 31 USING STOGROUP DBDST1 ,
      PART 32 USING STOGROUP DBDST1)
    LOCKSIZE ANY
```

```
   LOCKMAX SYSTEM
   LOCKPART YES
   CLOSE NO
   COMPRESS NO;
  COMMIT;
CREATE TABLE ASCBENCH1 (
KEYCOL CHAR(12) NOT NULL ,
RAND INTEGER NOT NULL ,
LOOKUPKEY INTEGER NOT NULL,
CUSTNAME CHAR(15) NOT NULL,
CUSTCODE CHAR(15) NOT NULL,
CUSTBALANCE INTEGER NOT NULL,
CUSTLIMIT INTEGER NOT NULL,
CUSTDOB CHAR(10) NOT NULL,
CUSTAGE INTEGER NOT NULL,
CUSTADDRESS CHAR(10) NOT NULL,
CUSTSTREET CHAR(20) NOT NULL,
CUSTSTATE CHAR(15) NOT NULL,
CUSTZIP CHAR(10) NOT NULL,
CUSTCOUNTRY CHAR(10) NOT NULL,
SOCIALSECURITYNO INTEGER NOT NULL,
CUSTSTARTDATE CHAR(10) NOT NULL,
CUSTRENEWDATE CHAR(10) NOT NULL,
CHAR01 CHAR(2) NOT NULL,
CHAR02 CHAR(2) NOT NULL,
CHAR03 CHAR(2) NOT NULL,
CHAR04 CHAR(2) NOT NULL,
CHAR05 CHAR(2) NOT NULL,
CHAR06 CHAR(2) NOT NULL,
CHAR07 CHAR(2) NOT NULL,
CHAR08 CHAR(2) NOT NULL,
CHAR09 CHAR(2) NOT NULL,
CHAR10 CHAR(2) NOT NULL,
CHAR11 CHAR(4) NOT NULL,
CHAR12 CHAR(4) NOT NULL,
CHAR13 CHAR(4) NOT NULL,
CHAR14 CHAR(4) NOT NULL,
CHAR15 CHAR(4) NOT NULL,
CHAR16 CHAR(4) NOT NULL,
CHAR17 CHAR(4) NOT NULL,
CHAR18 CHAR(4) NOT NULL,
CHAR19 CHAR(4) NOT NULL,
CHAR20 CHAR(4) NOT NULL,
CHAR21 CHAR(6) NOT NULL,
CHAR22 CHAR(6) NOT NULL,
CHAR23 CHAR(6) NOT NULL,
CHAR24 CHAR(6) NOT NULL,
CHAR25 CHAR(6) NOT NULL,
CHAR26 CHAR(6) NOT NULL,
CHAR27 CHAR(6) NOT NULL,
CHAR28 CHAR(6) NOT NULL,
CHAR29NEW CHAR(6) NOT NULL,
CHAR30NEW CHAR(3) NOT NULL,
CHAR31NEW CHAR(3) NOT NULL,
INTEGER01 INTEGER NOT NULL,
INTEGER02 INTEGER NOT NULL,
INTEGER03 INTEGER NOT NULL,
INTEGER04 INTEGER NOT NULL,
INTEGER05 INTEGER NOT NULL,
INTEGER06 INTEGER NOT NULL,
INTEGER07 INTEGER NOT NULL,
INTEGER08 INTEGER NOT NULL,
INTEGER09 INTEGER NOT NULL,
INTEGER10 INTEGER NOT NULL,
INTEGER11 INTEGER NOT NULL,
INTEGER12 INTEGER NOT NULL,
INTEGER13 INTEGER NOT NULL,
INTEGER14 INTEGER NOT NULL,
INTEGER15 INTEGER NOT NULL,
INTEGER16 INTEGER NOT NULL,
```

```
INTEGER17 INTEGER NOT NULL,
INTEGER18 INTEGER NOT NULL,
INTEGER19 INTEGER NOT NULL,
INTEGER20 INTEGER NOT NULL,
INTEGER21 INTEGER NOT NULL,
INTEGER22 INTEGER NOT NULL,
INTEGER23 INTEGER NOT NULL,
INTEGER24 INTEGER NOT NULL,
INTEGER25 INTEGER NOT NULL,
 INTEGER26 INTEGER NOT NULL,
 INTEGER27 INTEGER NOT NULL,
 INTEGER28 INTEGER NOT NULL,
 INTEGER29 INTEGER NOT NULL,
 INTEGER30 INTEGER NOT NULL,
 PACKED01 DECIMAL(1) NOT NULL,
 PACKED02 DECIMAL(3) NOT NULL,
 PACKED03 DECIMAL(5) NOT NULL,
 PACKED04 DECIMAL(7) NOT NULL,
 PACKED05 DECIMAL(9) NOT NULL,
 PACKED06 DECIMAL(11) NOT NULL,
 PACKED07 DECIMAL(13) NOT NULL,
 PACKED08 DECIMAL(15) NOT NULL,
 PACKED09 DECIMAL(1) NOT NULL,
 PACKED10 DECIMAL(3) NOT NULL,
 PACKED11 DECIMAL(5) NOT NULL,
 PACKED12 DECIMAL(7) NOT NULL,
 PACKED13 DECIMAL(9) NOT NULL,
 PACKED14 DECIMAL(11) NOT NULL,
 PACKED15 DECIMAL(13) NOT NULL,
 PACKED16 DECIMAL(15) NOT NULL,
 PACKED17C CHAR(17) NOT NULL,
 PACKED18C CHAR(18) NOT NULL,
 PACKED19C CHAR(19) NOT NULL,
 PACKED20C CHAR(20) NOT NULL)
   IN DBDSTAGE.TSBENCH1
   AUDIT NONE
   DATA CAPTURE NONE;
 GRANT SELECT ON TABLE DSTAGE.ASCBENCH1 TO PUBLIC;
 COMMIT;
 CREATE UNIQUE INDEX DSTAGE.XASCBENCH1
   ON DSTAGE.ASCBENCH1
    (KEYCOL              ASC )
   USING STOGROUP DBDST2
   PRIQTY 864000 SECQTY 72000
   FREEPAGE 0 PCTFREE 0
   CLUSTER
   ( PART 1 VALUES('000063045392') ,
     PART 2 VALUES('000126090784') ,
     PART 3 VALUES('000189136176') ,
     PART 4 VALUES('000252181568') ,
     PART 5 VALUES('000315226960') ,
     PART 6 VALUES('000378272352') ,
     PART 7 VALUES('000441317744') ,
     PART 8 VALUES('000504363136') ,
     PART 9 VALUES('000567408528') ,
     PART 10 VALUES('000630453920') ,
     PART 11 VALUES('000693499312') ,
     PART 12 VALUES('000756544704') ,
     PART 13 VALUES('000819590096') ,
     PART 14 VALUES('000882635488') ,
     PART 15 VALUES('000945680880') ,
     PART 16 VALUES('001008726272') ,
     PART 17 VALUES('001071771664') ,
     PART 18 VALUES('001134817056') ,
     PART 19 VALUES('001197862448') ,
     PART 20 VALUES('001260907840') ,
     PART 21 VALUES('001323953232') ,
     PART 22 VALUES('001386998624') ,
     PART 23 VALUES('001450044016') ,
     PART 24 VALUES('001513089408') ,
```

```
    PART 25 VALUES('001576134800') ,
    PART 26  VALUES('001639180192') ,
    PART 27  VALUES('001702225584') ,
    PART 28  VALUES('001765270976') ,
    PART 29  VALUES('001828316368') ,
    PART 30  VALUES('001891361760') ,
    PART 31  VALUES('001954407152') ,
    PART 32  VALUES('002017452544') )
  CLOSE NO
  COPY NO ;
COMMIT;
```

Appendix B – Sample JCL to run the DS/390 COBOL program

```
//RUN001 JOB ,'DATASTAGE',CLASS=A,MSGCLASS=X,
//     REGION=0M,LINES=50000,NOTIFY=&SYSUID
//*********************************************************************
//****  OS390_DELETEFILE
//*********************************************************************
//STEPDEL EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  LISTCAT ENTRIES(M9LRF1.LOAD.FILE001)
  IF LASTCC = 0 -
    THEN -
      DELETE M9LRF1.LOAD.FILE001
    ELSE -
      SET MAXCC = 0
/*
//*********************************************************************
//****  OS390_RUN
//*********************************************************************
//RUN     EXEC PGM=BENCH2,TIME=1440
//STEPLIB  DD  DISP=SHR,DSN=DSTAGE.DS390.LOAD
//REPORT   DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//RTLOUT   DD  SYSOUT=*
//*********************************************************************
//****  OS390_OLDFILE  (32 GB input stream)
//*********************************************************************
//DRIVER DD DISP=SHR,DSN=M9DRV1.DRIVER.FILE001,DCB=BUFNO=50
//       DD DISP=SHR,DSN=M9DRV1.DRIVER.FILE002,DCB=BUFNO=50
//*      DD DISP=SHR,DSN=M9DRV1.DRIVER.FILE003,DCB=BUFNO=50
//*      DD DISP=SHR,DSN=M9DRV1.DRIVER.FILE004,DCB=BUFNO=50
//*********************************************************************
//****  OS390_OLDFILE
//*********************************************************************
//LOOKUP DD DISP=SHR,DSN=DSTAGE.LOOKUP.FILE,DCB=BUFNO=50
//*********************************************************************
//****  OS390_NEWFILE
//*********************************************************************
//DB2LOAD DD DSN=M9LRF1.LOAD.FILE001,
//      DISP=(NEW,CATLG,DELETE),
//      UNIT=SYSDA,
//      SPACE=(CYL,(10000,3333),RLSE),
//      DCB=(LRECL=547,BLKSIZE=27897,RECFM=FB,BUFNO=50)
//*********************************************************************
//****  OS390_DB2LOAD
//*********************************************************************
//LOAD2 EXEC PGM=DSNUTILB,
//       COND=(4,LT),
//       REGION=4096K,
//       PARM='DBL1,RUN001'
//STEPLIB  DD DISP=SHR,DSN=DSN710.SDSNLOAD
//SYSREC   DD DISP=SHR,DSN=M9LRF1.LOAD.FILE001
//SYSPRINT DD SYSOUT=*
//UTPRINT  DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,DSN=M9LRF1.RUN001.SYSUT1,
//         SPACE=(CYL,(1600,50),RLSE),DISP=(,DELETE,DELETE)
//SORTOUT  DD UNIT=SYSDA,
//         SPACE=(TRK,(10,10),RLSE)
//SORTWK01 DD UNIT=SYSDA,
//         SPACE=(TRK,(10,10),RLSE)
//SORTWK02 DD UNIT=SYSDA,
//         SPACE=(TRK,(10,10),RLSE)
//SORTWK03 DD UNIT=SYSDA,
//         SPACE=(TRK,(10,10),RLSE)
//SYSIN    DD *
LOAD DATA LOG NO
   INTO TABLE DSTAGE.ASCBENCH1 PART 1
```

```
(
KEYCOL POSITION(1) CHARACTER(12),
RAND POSITION(13) INTEGER,
LOOKUPKEY POSITION(17) INTEGER,
CUSTNAME POSITION(21) CHARACTER(15),
CUSTCODE POSITION(36) CHARACTER(15),
CUSTBALANCE POSITION(51) INTEGER,
CUSTLIMIT POSITION(55) INTEGER,
CUSTDOB POSITION(59) CHARACTER(10),
CUSTAGE POSITION(69) INTEGER,
CUSTADDRESS POSITION(73) CHARACTER(10),
CUSTSTREET POSITION(83) CHARACTER(20),
CUSTSTATE POSITION(103) CHARACTER(15),
CUSTZIP POSITION(118) CHARACTER(10),
CUSTCOUNTRY POSITION(128) CHARACTER(10),
SOCIALSECURITYNO POSITION(138) INTEGER,
CUSTSTARTDATE POSITION(142) CHARACTER(10),
CUSTRENEWDATE POSITION(152) CHARACTER(10),
CHAR01 POSITION(162) CHARACTER(2),
CHAR02 POSITION(164) CHARACTER(2),
CHAR03 POSITION(166) CHARACTER(2),
CHAR04 POSITION(168) CHARACTER(2),
CHAR05 POSITION(170) CHARACTER(2),
CHAR06 POSITION(172) CHARACTER(2),
CHAR07 POSITION(174) CHARACTER(2),
CHAR08 POSITION(176) CHARACTER(2),
CHAR09 POSITION(178) CHARACTER(2),
CHAR10 POSITION(180) CHARACTER(2),
CHAR11 POSITION(182) CHARACTER(4),
CHAR12 POSITION(186) CHARACTER(4),
CHAR13 POSITION(190) CHARACTER(4),
CHAR14 POSITION(194) CHARACTER(4),
CHAR15 POSITION(198) CHARACTER(4),
CHAR16 POSITION(202) CHARACTER(4),
CHAR17 POSITION(206) CHARACTER(4),
CHAR18 POSITION(210) CHARACTER(4),
CHAR19 POSITION(214) CHARACTER(4),
CHAR20 POSITION(218) CHARACTER(4),
CHAR21 POSITION(222) CHARACTER(6),
CHAR22 POSITION(228) CHARACTER(6),
CHAR23 POSITION(234) CHARACTER(6),
CHAR24 POSITION(240) CHARACTER(6),
CHAR25 POSITION(246) CHARACTER(6),
CHAR26 POSITION(252) CHARACTER(6),
CHAR27 POSITION(258) CHARACTER(6),
CHAR28 POSITION(264) CHARACTER(6),
CHAR29NEW POSITION(270) CHARACTER(6),
CHAR30NEW POSITION(276) CHARACTER(3),
CHAR31NEW POSITION(279) CHARACTER(3),
INTEGER01 POSITION(282) INTEGER,
INTEGER02 POSITION(286) INTEGER,
INTEGER03 POSITION(290) INTEGER,
INTEGER04 POSITION(294) INTEGER,
INTEGER05 POSITION(298) INTEGER,
INTEGER06 POSITION(302) INTEGER,
INTEGER07 POSITION(306) INTEGER,
INTEGER08 POSITION(310) INTEGER,
INTEGER09 POSITION(314) INTEGER,
INTEGER10 POSITION(318) INTEGER,
INTEGER11 POSITION(322) INTEGER,
INTEGER12 POSITION(326) INTEGER,
INTEGER13 POSITION(330) INTEGER,
INTEGER14 POSITION(334) INTEGER,
INTEGER15 POSITION(338) INTEGER,
INTEGER16 POSITION(342) INTEGER,
INTEGER17 POSITION(346) INTEGER,
INTEGER18 POSITION(350) INTEGER,
INTEGER19 POSITION(354) INTEGER,
INTEGER20 POSITION(358) INTEGER,
INTEGER21 POSITION(362) INTEGER,
```

```
   INTEGER22 POSITION(366) INTEGER,
   INTEGER23 POSITION(370) INTEGER,
   INTEGER24 POSITION(374) INTEGER,
   INTEGER25 POSITION(378) INTEGER,
   INTEGER26 POSITION(382) INTEGER,
   INTEGER27 POSITION(386) INTEGER,
   INTEGER28 POSITION(390) INTEGER,
   INTEGER29 POSITION(394) INTEGER,
   INTEGER30 POSITION(398) INTEGER,
   PACKED01 POSITION(402) DECIMAL PACKED,
   PACKED02 POSITION(403) DECIMAL PACKED,
   PACKED03 POSITION(405) DECIMAL PACKED,
   PACKED04 POSITION(408) DECIMAL PACKED,
   PACKED05 POSITION(412) DECIMAL PACKED,
   PACKED06 POSITION(417) DECIMAL PACKED,
   PACKED07 POSITION(423) DECIMAL PACKED,
   PACKED08 POSITION(430) DECIMAL PACKED,
   PACKED09 POSITION(438) DECIMAL PACKED,
   PACKED10 POSITION(439) DECIMAL PACKED,
   PACKED11 POSITION(441) DECIMAL PACKED,
   PACKED12 POSITION(444) DECIMAL PACKED,
   PACKED13 POSITION(448) DECIMAL PACKED,
   PACKED14 POSITION(453) DECIMAL PACKED,
   PACKED15 POSITION(459) DECIMAL PACKED,
   PACKED16 POSITION(466) DECIMAL PACKED,
   PACKED17C POSITION(474) CHARACTER(17),
   PACKED18C POSITION(491) CHARACTER(18),
   PACKED19C POSITION(509) CHARACTER(19),
   PACKED20C POSITION(528) CHARACTER(20)
   )
/*
```

Changes to JCL to implement Batch Pipes

```
//BMBP32A JOB ,'DATASTAGE',CLASS=A,MSGCLASS=X,
//     REGION=0M,LINES=50000,NOTIFY=&SYSUID
//*******************************************************************
//****  OS390_RUN
//*******************************************************************
//*******************************************************************
SAME AS IN ORIGINAL JCL
//*******************************************************************
//*******************************************************************
//****  OS390_NEWFILE
//*******************************************************************
//DB2LOAD DD DSN=M9LRF1.BPLOAD.FILE001,SUBSYS=BP01,                  ←- Changed
//        DCB=(LRECL=547,RECFM=FB,BUFNO=254)                         ←- Changed
//*
//********  SECOND JOB  *************
//*
//BMBP32B JOB ,'DATASTAGE',CLASS=A,MSGCLASS=X,
//     REGION=0M,LINES=50000,NOTIFY=&SYSUID
//*******************************************************************
//****  OS390_DB2LOAD
//*******************************************************************
//LOAD1 EXEC PGM=DSNUTILB,
//       COND=(4,LT),
//       REGION=4096K,
//       PARM='DBL1,RUN001'
//STEPLIB  DD DISP=SHR,DSN=DSN710.SDSNLOAD
//SYSREC   DD DISP=SHR,DSN=M9LRF1.BPLOAD.FILE001,SUBSYS=BP01,        ←- Changed
//        DCB=(LRECL=547,RECFM=FB,BUFNO=254)                         ←- Changed
//*******************************************************************
SAME AS IN ORIGINAL JCL
//*******************************************************************
```