Meikel Poess, Tilmann Rabl, Michael Frank, and Manuel Danisch

A PDGF Implementation for TPC-H

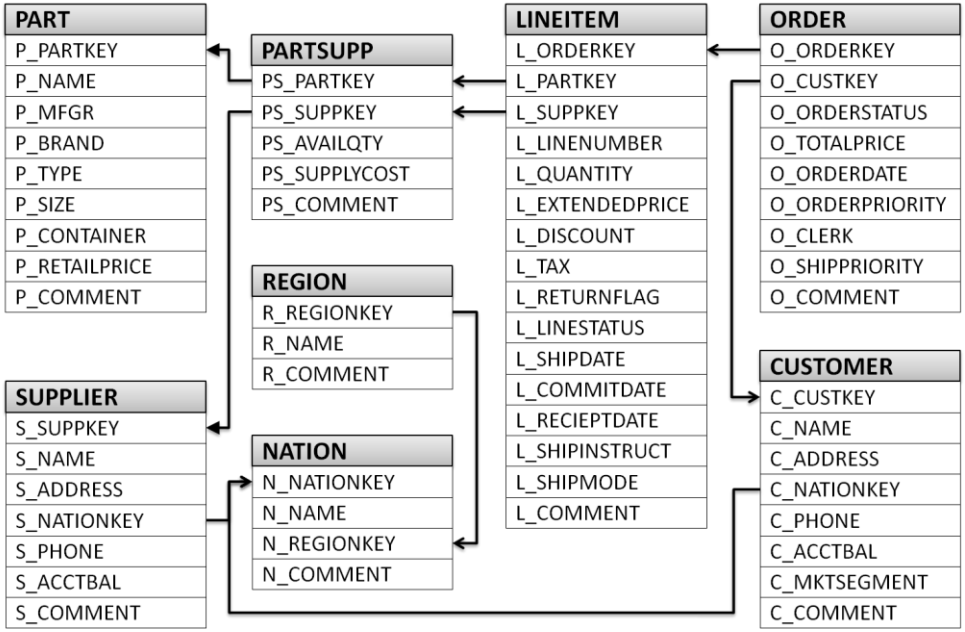
Third TPC Technology Conference on Performance Evaluation & Benchmarking
August 29, 2011

Agenda

- Motivation
- Parallel Data Generation Framework PDGF
- PDGF Implementation for TPC-H
- Verification of PDGF's TPC-H implementation
- Conclusion

TPC-H - Overview

- Introduced in 1999 (based on TPC-D)
- 182 benchmark publications and counting
- 8 Tables
- 61 columns
- Third normal form
- Scaled by SF
  - SF = 1 … 100,000
- Needs to generate data quickly



DBGEN

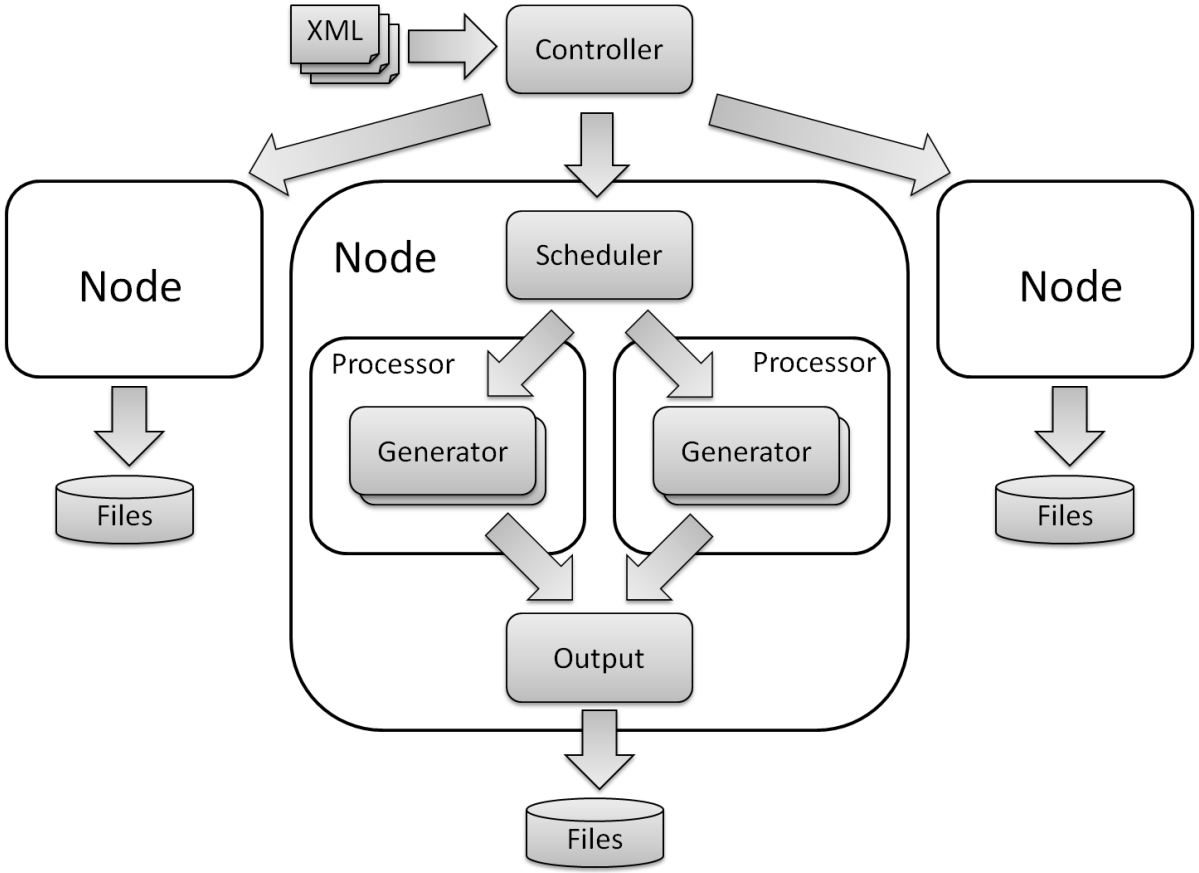
- Is TPC-H's current data generator
- Inherited from TPC-D
- Implemented in ANSI-C
- Ported to 20 different platforms

Parallel Data Generator Framework

- Designed at the University of Passau by Tilmann Rabl
- Was first presented at TPCTC 2010
- Is a generic data generator written in Java
- Can be configured to generate any RDBMS schema
- Can be configured to generate most data types and distributions
  - Numbers, strings, dates, etc
  - Uniform, Gaussian, etc.
- Is extensible
- Generates data in parallel (within the same address space and across address spaces)

PDGF – Architectural Overview

- Parallel Data Generation Framework (PDGF)

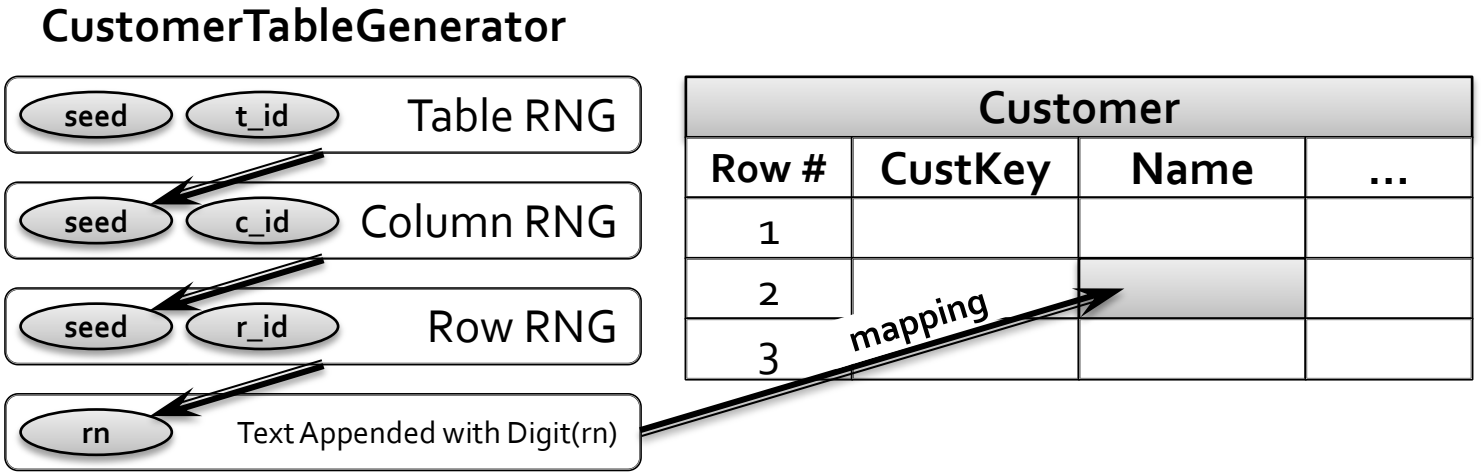


Configuration and Implementation

- XML files for configuration
  - Reflects SQL schema
    - Tables
    - Attributes
  - Seed
  - Size
  - Scale factor
  - Output
- Plug-in mechanism
  - Generators
  - Distributions
  - Output

```
<schema>
…
  <tables>
    <table name="ORDERS">
      <size>1500000</size>
      <fields>
        <field name="O_ORDERKEY">
          <type>java.sql.Types.INTEGER</type>
          <generator name="O_OrderKey">
          </generator>
        </field>
…
```

PDGF – Seeding Strategy



- Hierarchical seeding
- Seeds can be cached
- Generation of n-th value with n-th random number
- Easy reference generation
- Embarrassingly parallel

Comparison DBGEN and PDGF

DBGEN

- Contains platform specific implementations → is prone to platform specific bugs
- Needs to be compiled by each vendor on each platform
- Only implements the TPC-H schema
- Has values and data distributions hardcoded
- Generates data in parallel within an address space and across address spaces

PDGF

- Is implemented in Java →Platform independent
- Can be shipped in byte code
- Can implement any RDBMS schema, including TPC-H
- Separates schema and data definition from core data generator
- Generates data in parallel within an address space and across address spaces

… but can PDGF be used for TPC-H ?

TPC-H – Data Specification

- Clause 4: Scaling and database population:
  - Row counts
  - Detailed data specification for all columns
    - 8 Data primitives
    - 15 Several special cases

Primitive	#Columns	Example Column	Sample output
Unique Value[min,max]	7	O_ORDERKEY unique within [SF * 1,500,000]	12398709
Date[min,max]	4	O_Orderdate=Date[1992-01-01,1998-08-02]	1995-05-26
Phone Number	2	S_Phone=Phone Number	16-421-927-9442
Random String [instructions]	6	L_Shipinstruct=Random String [instructions]	TAKE BACK RETURN
Random Value [min,max]	12	S_Nationkey=Random Value [0,24]	23
Random v-string	2	S_Address=Random v-String [10,40]	vs50U4?e5i
Text Append with Digit	5	S_Name=Text Appended with Digit ["Supplier", S_Suppkey]	Supplier5628
Text String	8	PS_Comment=Text String [49,198]	dependencies beyo

TPC-H – Data Specification

- 15 Special Cases
  - Constants, e.g.
    - O_SHIPPRIORITY set to 0
  - Intra row dependencies , e.g.
    - L_LINESTATUS set the following value: "O" if L_SHIPDATE > CURRENTDATE "F" otherwise.
  - Intra table dependencies , e.g.
    - O_ORDERSTATUS set to the following value:
      - "F" if all lineitems of this order have L_LINESTATUS set to "F".
      - "O" if all lineitems of this order have L_LINESTATUS set to "O".
      - "P" otherwise.
  - Intra table dependencies , e.g.
    - L_EXTENDEDPRICE = L_QUANTITY * P_RETAILPRICE (where L_PARTKEY=P_PARTKEY)

Date

- Uniformly distributed within start and end date
- PDGF uses millisecond representation
  - Standard generator, uses Java date formating
- Generation
  - Pick random number between start and end date
- Special cases
  - L_Shipdate: 121 days after O_Orderdate
  - Special generator: Calculate reference, add 121 days
  - Similarly: L_Receiptdate, L_Commitdate

```
<field name="O_ORDERDATE">
  <type>java.sql.Types.DATE</type>
  <generator name="DateGenerator">
    <startDate>1992-01-01</startDate>
    <endDate>1998-08-02</endDate>
  </generator>
</field>
```

O_Totalprice

- Inter-table dependencies
  - Calculated over all lineitems with same L_Orderkey
  - *sum(L_Extendedprice*(1+L_Tax)*(1-L_Discount))*
- L_Extendedprice
  - *L_Quantity * P_Retailprice* where *L_Partkey = P_Partkey*
- Solved with PDGF reference generation

```
<field name="O_TOTALPRICE">
  <type>java.sql.Types.DECIMAL</type>
  <generator name="O_TotalPrice">
  </generator>
</field>
```

Verifying TPC-H Data from PDGF

- Mandatory requirements:
  - Row counts: Need to match exactly according to SF
    → Simple row count
  - Derived fields: Need to match exactly according to specification
    → Possibly require complex joins
  - All other fields: Need to be statistically equivalent
    → We use coefficient of variation (CoV)
- Our Approach is to use compliance queries written in SQL

Table Cardinalities

- Cardinalities for Orders, Customer, Supplier, Part, Partsupp, Nation, Region are specified in the TPC-H specification

  → Can be checked with simple "select count(*)"

Table	Table cardinalities @ SF=100		
	Specification	DBgen	PDGF
Orders	150 Million	150 Million	150 Million
Customer	15 Million	15 Million	15 Million
Supplier	1 Million	1 Million	1 Million
Part	20 Million	20 Million	20 Million
Partsupp	80 Million	80 Million	80 Million
Nation	25	25	25
Region	5	5	5

Cardinality Lineitem Table

- Cardinality of Lineitem is defined as:
  - For each row in the Orders table, a random number of rows within [1 .. 7] exist in Lineitem
  - Need to test three characteristics:
    1. Join Frequency
       - 1 through 7
    2. Coefficient of the frequency distribution
       - 0.000197 for DBgen and 0.000002 for PDGF
    3. Row count
       - DBGEN=600,037,902 rows and PDGF 600,000,000
       - 0.006317% difference

Verifying Date Columns

```
SELECT MIN(O_Orderdate)
      ,MAX(O_Orderdate)
      ,count(distinct O_Orderdate)
FROM Orders;
SELECT STDDEV(c)/AVG(c)
FROM (SELECT O_Orderdate,count(*) c
      FROM Orders
      GROUP BY O_Orderdate);
```

Column	CoV of dates		Date Range DBgen			Date Range PDGF		
	DBgen	PDGF	Min	Max	#distinct	Min	Max	#distinct
O_Orderdate	0.00388	0.00398	1992-01-01	1998-08-02	2406	1992-01-01	1998-08-02	2406
L_Shipdate	0.17970	0.17969	1992-01-02	1998-12-01	2526	1992-01-02	1998-12-01	2526
L_Commitdate	0.12762	0.12763	1992-01-31	1998-10-31	2466	1992-01-31	1998-10-31	2466
L_Receiptdate	0.2088	0.20887	1992-01-	1998-12-31	2555	1992-01-	1998-12-	2555

Verifying O_TOTALPRICE

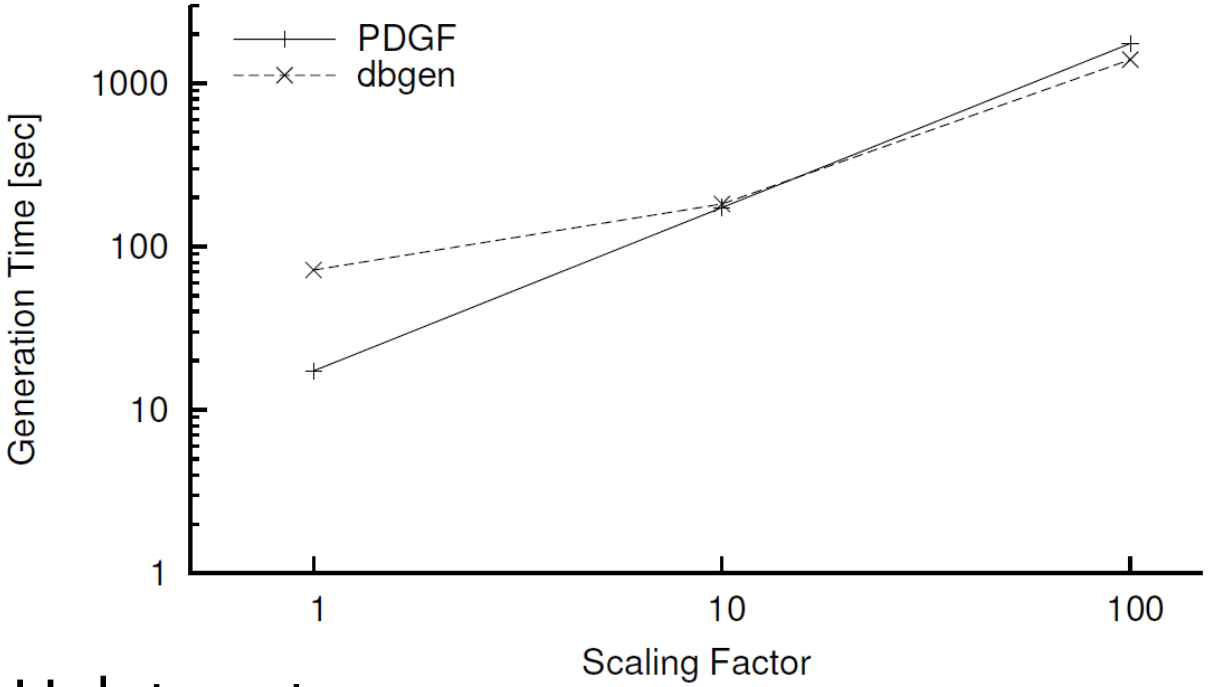
```
SELECT COUNT(*)
FROM(SELECT O1.O_Orderkey OK,
            SUM(L1.L_Extendedprice*(1+L1.L_Tax)*(1-L1.L_Discount)) TP
     FROM Lineitem L1,Orders O1
     WHERE L1.L_Orderkey=O1.O_Orderkey
     GROUP BY O1.O_Orderkey),Orders O2
WHERE OK<>O2.O_Orderkey And O2.O_Totalprice<>TP;
```

This query returns zero rows if the data is correct

Verification Summary

- DBGEN shows a wide range for the CoV of various columns:
  - E.g. CoV of the distribution of lineitem to orders is 0.000197 while the CoV of L_Partkey is 0.15503.
  - It is up to the TPC to decide whether these CoV are specification conforming
- For our comparison it is only important whether the data PDGF generates has the same or better CoV
- For most columns the CoV of PDGF data is better than that of DBGEN data
- For few columns DBGEN generates data with a better CoV:
  - E.g. Ps_Supplycost shows a CoV of 0.31573 with PDGF and 0.03469 with DBGEN
- Detailed data is in the paper

PDGF vs DBGEN



- TPC-H data set
  - Data sizes: 1 GB, 10 GB, 100 GB
- Single node (8 cores)
  - 2 Intel Xeon QuadCore processors, 16GB RAM

Conclusion

- Demonstrated that PDGF is a viable alternative to DBGEN
- PDGF has many advantages over DBGEN
  - Generic
  - Java based
- Could be used as THE data generation framework in the TPC

Future Work

- Generate updates
- Extend framework to generate queries
- Analyze and potentially fix mismatch with TPC-H data
- Implement other TPC benchmarks