Why Did Jim Gray Win the Turing Award?

Michael Stonebraker
CSAIL, MIT
77 Massachusetts Avenue
Cambridge, Ma 02139
603-714-4451
stonebraker@csail.mit.edu

Abstract

This short paper is intended to describe for the layman why Jim Gray won so many awards, culminating in his being selected to receive the 1998 ACM Turing Award, arguably the "Nobel Prize of Computer Science". It briefly summarizes his main contributions to our field.

1. Transactions and More Transactions

After Ted Codd's pioneering paper appeared in CACM in June of 1970, there was an immediate debate between:

The relational advocates who argued that a simple data model (tables) and a high level language (at the time Codd's Data Language Alpha or relational algebra; now SQL) were obviously good, and should form the basis of a sensible DBMS architecture

The IMS and CODASYL advocates, who argued that it was impossible to implement relational query languages efficiently. Moreover, no real programmers could possibly understand Codd's query languages

It was obvious to many researchers that the next required steps were to:

1) Specify more user-friendly relational query languages

2) Prove that relational DBMSs were practical

The first task led to several new query languages, of which SQL is the one that won in the marketplace (largely because of the "throw weight" of IBM). The second task led to a collection of implementations, of which Ingres (at Berkeley) and System R (at IBM Research) were the most fully developed.

Jim was one of the researchers working on System R.

Two of the messy issues that had to be dealt with in any implementation were:

crash recovery and

concurrency control.

Obviously, a DBMS should never lose customer data, regardless of what sort of failure occurs. However, what exactly does this mean and how should it be implemented? In addition, most DBMSs must deal with parallel update from multiple users or applications. For example, one user might be moving all shoe department employees into the toy department while a simultaneous user might be giving a 10% raise to all toy department employees. Clearly, the collection of individuals specified by the first user is being altered by the second user. In such a situation, one must decide what semantics to enforce and how to do so efficiently.

Jim wrote a pioneering paper in 1976 and followed this up with a book in the mid 1980s on this topic. He is largely responsible for the following (very simple in retrospect – but revolutionary at the time) ideas.

One should divide DBMS activity into units of work, called **transactions**. A transaction consists of one or more statements in SQL (or whatever interaction language is supported) interspersed with code in a general purpose programming language. For example, a transaction might consist of moving $100 from account A to account B. In SQL (and most other interaction languages), this requires two statements, one to decrement account A and one to increment account B.

Each transaction must have the following properties:

Atomic: either the entire transaction happens or none of it happens. I.e. it is illegal to have the decrement happen unless the paired increment also happens. Hence, transactions move the data base from one consistent state to another.

Consistent: The data base is free to define a collection of **integrity constraints**, that define legal data base states. One such requirement might be that account balances are non-negative. Any transaction that makes an update which violates an integrity constraint must be aborted. Hence, it is illegal to execute a transaction that produces an inconsistent DBMS state.

Isolation: This requirement means that parallel transactions cannot see the intermediate states of other transactions. In other words, the outcome of this collection must be the same as the collection run in some serial order, one after the other. Any other outcome is an inconsistent state. There is no requirement to obey any specific serial order, just a requirement to obey some **serial** ordering. This requirement defines legal data bases states when a collection of parallel transactions are run,

Durable: In the event of a failure, there are only two possible outcomes. Either a transaction "happened", i.e. it is committed or it did not happen, i.e. it is aborted. If the user was notified that the transaction committed, then the DBMS agrees that it cannot develop a case of amnesia. Hence, the effects of the transaction can never be lost, regardless of what failures might occur.

Together, these are called the "ACID properties". Supporting these properties efficiently is a deep intellectual topic, about which much has been written over the last quarter of a century. For example, one simple scheme is to "lock" all objects a transaction touches and hold all locks until the transaction ends. Every time a transaction makes an update, a "log record" is written holding both the "before image" of the object as well as the "after image". If the transaction must be undone, then the before image is used to "rewind" the database. If the effect of a committed transaction is lost, because of a storage failure, then the

after image is used to restore the effects of the committed transaction.

Working out the properties of transactions and then constructing efficient implementation schemes was the major contribution of Jim Gray in the 1970's and early 1980's. For this pioneering effort, he received the Turing Award in 1998.

In closing, I would like to mention that Jim had three characteristics that I truly admire. First, he was an intellectual sponge. He read voraciously in many areas of Computer Science, and seemed to know "everything about everything". Second, he was always willing to spend time discussing new ideas, and would freely give his perspective on other researchers' thoughts. As such, he mentored and helped many, many people, including me, for which I am very grateful. Third, he is one of the smartest people I have ever known. This combination of intellectual curiosity, willingness to help others, and raw intellectual ability is rarely found, and made Jim a true giant in our field.