A distributed system is:

A collection of independent computers that appears to its users as a single coherent system.
A distributed system organized as middleware. Note that the middleware layer extends over multiple machines.
## Transparency in a Distributed System

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>Migration</td>
<td></td>
</tr>
<tr>
<td>Relocation</td>
<td></td>
</tr>
<tr>
<td>Replication</td>
<td></td>
</tr>
<tr>
<td>Concurrency</td>
<td></td>
</tr>
<tr>
<td>Failure</td>
<td></td>
</tr>
<tr>
<td>Persistence</td>
<td></td>
</tr>
</tbody>
</table>

Different forms of transparency in a distributed system.
## Scalability Problems

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized services</td>
<td>A single server for all users</td>
</tr>
<tr>
<td>Centralized data</td>
<td>A single on-line telephone book</td>
</tr>
<tr>
<td>Centralized algorithms</td>
<td>Doing routing based on complete information</td>
</tr>
</tbody>
</table>

Examples of scalability limitations.
The difference between letting:

a) a server or

b) a client check forms as they are being filled
An example of dividing the DNS name space into zones.
Lessons from Giant-Scale Services

Eric Brewer
UC Berkeley and Inktomi
IEEE Internet Computing July/August 2001
“Giant-Scale” Services

- Key real-world challenges
  - High availability
  - Evolution
  - Growth
Advantages of Giant-Scale Services

- Access anywhere, anytime
- Availability via multiple devices
- Groupware support
- Lower overall cost
- Simplified service updates
Figure 1. The basic model for giant-scale services. Clients connect via the Internet and then go through a load manager that hides down nodes and balances traffic.
Assumptions

- Service provider has limited control over clients/network
- Queries drive the service
- Read-only queries greatly outnumber updates
Google Data Centers

Brian Nettles/The Post and Courier
Load Management

- Round-Robin DNS
- “Layer 4” switch
- “Layer 7” switch
Comparison

Figure 2. A simple Web farm. Round-robin DNS assigns different servers to different clients to achieve simple load balancing. Persistent data is fully replicated and thus all nodes are identical and can handle all queries.

Figure 3. Search engine cluster. The service provides support to other programs (Web servers) rather than directly to end users. These programs connect via layer-4 switches that balance load and hide faults. Persistent data is partitioned across the servers, which increases aggregate capacity but implies there is some data loss when a server is down. A backplane allows all nodes to access all data.
Availability Metrics

- **Uptime**
  \[ \text{Uptime} = \frac{\text{MTBF} - \text{MTTR}}{\text{MTBF}} \]

- **Mean-time-between-failure (MTBF)**

- **Mean-time-to-repair (MTTR)**

- **Yield** = queries completed/queried offered

- **Harvest** = data available/complete data
DQ Principle

- Data per query x queries per second -> constant
Replication vs Partition
Graceful Degradation

- Peak-to-avg ratio = 1.6:1 to 6:1
- Single-event bursts can generate far above-average traffic
- Some faults are not independent

- Explicit process of managing the effect of saturation
  - Cost-based access control
  - Priority or value-based access control
  - Reduced data freshness
Online Evolution and Growth

- “Internet time” - frequent product releases
- Maintenance and upgrades
  = controlled failures
  = “online evolution”

- Fast reboot
- Rolling upgrade
- Big flip
Lessons

- Get the basics right
- Decide on your availability metrics
- Focus on MTTR at least as much as MTBF
- Understand load redirection during faults
- Graceful degradation
- Use DQ analysis on all upgrades
- Automate upgrades as much as possible