On Utilization of Contributory Storage in Desktop Grids

Chreston Miller, Ali R. Butt, and Patrick Butler

Department of Computer Science

Contributory Storage: Cheap Storage using Shared Resources

- Distributed setup with many participants
- Nodes contribute storage space for sharing
- Create a uniform global storage space
- Typically supports decentralized store/lookup

- Many systems build upon this idea
  - PAST, CFS, OceanStore, Kosha, LOCKSS,…

Goal: Use of Contributory Storage in Scientific Computing

- Advantages:
  - Provides economical storage with large capacity
  - Supports parallel access to distributed resources

- Challenges:
  - Limited individual file sizes
  - Unreliable and transient participants

  → Simple replication or file splitting is likely not to work

Need for techniques to use shared storage in scientific computing

Our Contribution: PeerStripe Reliable Shared Storage

- Utilizes storage contributed by peer nodes
- Adapts data striping to support large files
- Employs error coding for fault tolerance
- Leverages multicast for efficient replication
- Supports easy integration with applications

Outline

- Preamble
- End to our Means
- Evaluation Study
- Conclusion

Outline

- Preamble
  - Problem
  - Motivation
  - Our Contributions
  - Core Technologies
- End to our Means
- Evaluation Study
- Conclusion

Core Technologies: Structured Peer-to-Peer Networks

- Implement Distributed Hash Table abstraction
- Facilitate decentralized operation
- Provide self-organization of participants
- Systems based on these networks provide:
  - Mobility and location transparency
  - Load-balancing
- We use Free Pastry substrate from Rice University and Microsoft

Core Technologies: Increasing Data Availability

- Erasure codes
  - Provide redundancy against failures
  - Incur less space overhead than replication
  - Advanced codes can withstand multiple failures

- Multicast communication protocol
  - Supports simultaneous messaging to many nodes
  - Can be leveraged for efficient replication

Outline

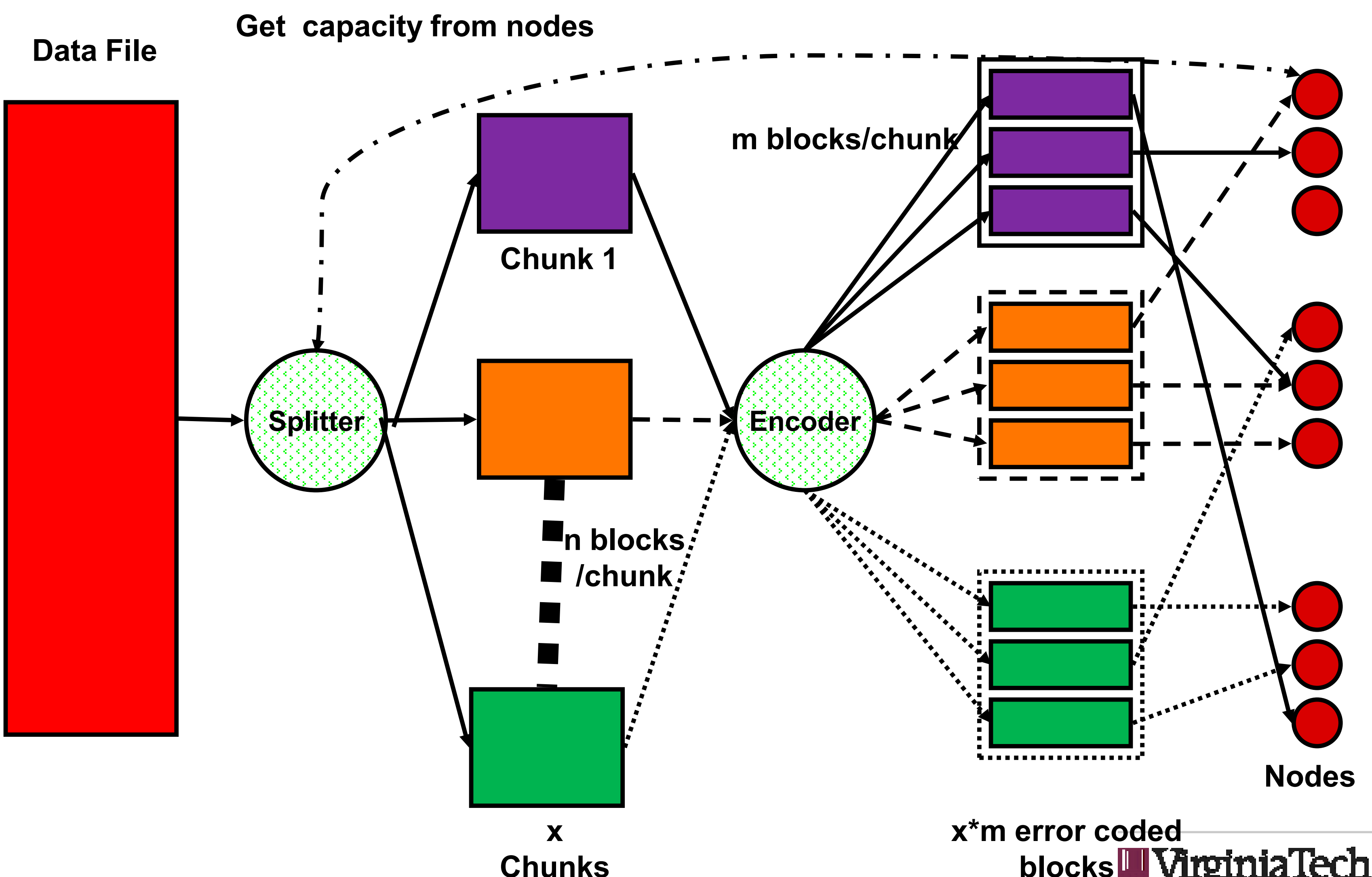
- Preamble
- End to our Means
    - Software Architecture
    - Splitting a file
    - Redundancy with multicast
    - Error coding
    - Interfacing with applications
- Experimental Study
- Conclusion

PeerStripe Software Tasks

1. Storing large files
   - Split file into different size chunks
   - Use DHT's to store chunks
2. Error coding chunks
   - Use online code to provide redundancy
3. Chunk replication
   - Replicate commonly used chunks
4. Interface with applications
   - Provide API's for applications to use

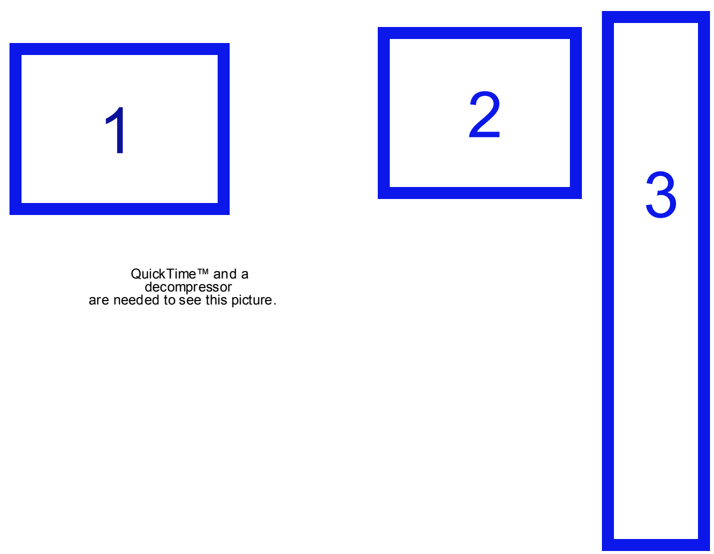
VirginiaTech
Invent the Future

Part 1: Splitting Files into Chunks



Part 2: Error Coding Chunks

- Each chunk is separately error coded
  1. A chunk is split into equal `n` size blocks
  2. The blocks are error coded into `m` encoded blocks
  3. Encoded blocks are inserted into the DHT



Investigation of Error Codes

- Error codes tested and used:
  - XOR code: Protect against single failures
  - Online code: Protect against multiple failures

+ Good redundancy with small space overhead

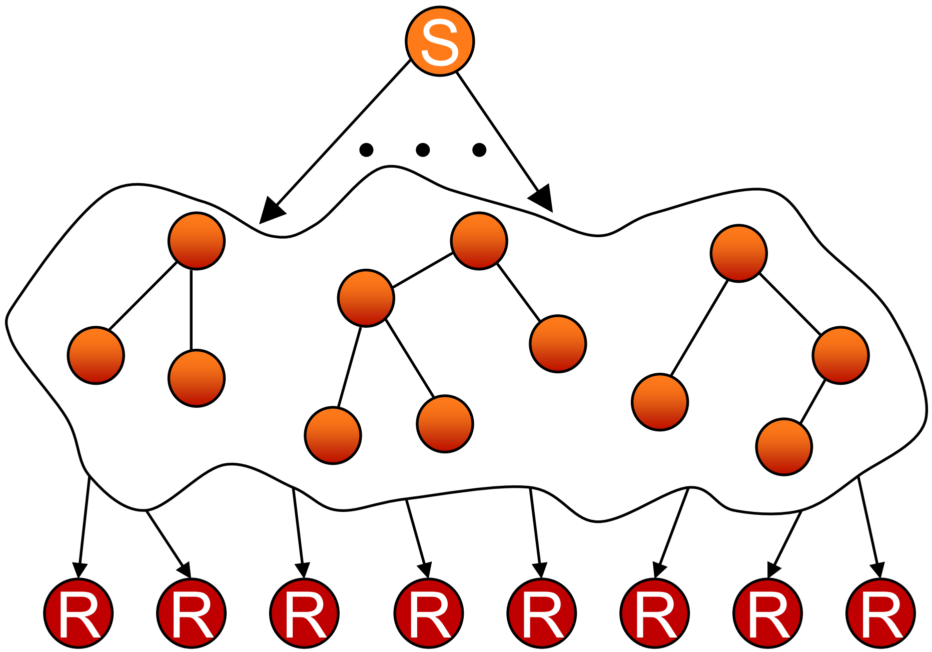
- Recovery may consume resources

Part 3: Multicast-based Replication

- Leverage multicast for efficient and fast data dissemination to multiple destinations
- Faster recovery at the cost of space
- Challenge: Creation of a multicast-tree from source to replica destinations

Creating a Multicast Tree

- Use greedy approach
  - Start from the source **S**
  - Using locality-aware DHT select random nodes close to **S** as first tier
  - Repeat selecting at each tier till replica location **R** is reached
- Employ standard multicast protocols, e.g. Bullet to push data from **S** to **R**



VirginiaTech Invent the Future

Part 4: Interfacing with Applications

- Modify applications to use direct calls to the PeerStripe API
  - Works well for new applications
- Link applications with an interposing library to redirect I/O
  - Transparent integration with existing applications

Outline

- Begin to our Means
- End to our Means
- Evaluation Study
  - Simulation
  - Real world
    - PlanetLab
    - Condor
- Conclusion

Evaluation: Overview

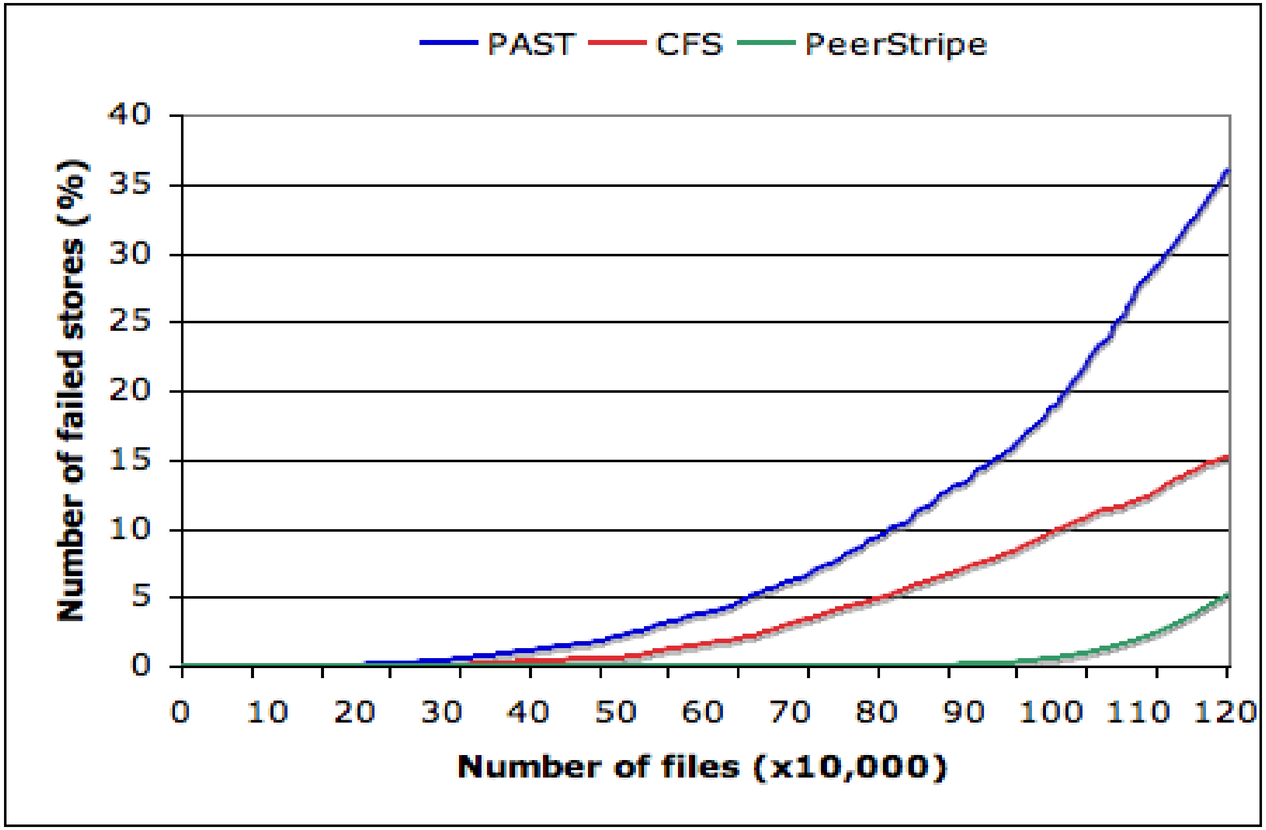
1. Simulation study:
   - Successful File Stores
   - Number and size of chunks created
   - System utilization (in terms of storage capacity)
   - File availability with error coding
   - Error code performance
   - Effects of participant churn
2. Design verification on PlanetLab
3. Integration with Condor desktop grid

Simulation Study Setup

- 10,000-node directly connected network
- Assigned node capacities with mean 45 GB and variance 10 GB

- File system trace of 1.2M files totaling 278.7 TB
- Compare with PAST and CFS storage systems

Number of Successful File Stores



- 7.0x improvement over PAST
- 2.9x improvement over CFS

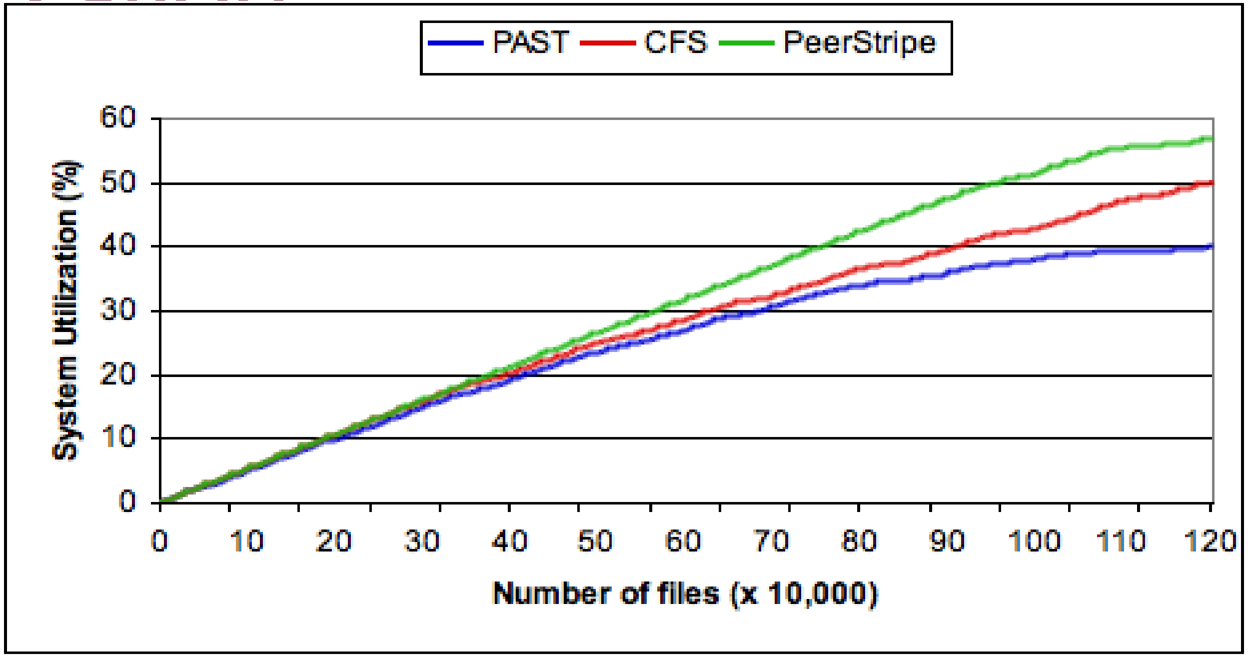
Number and Size of Chunks

- CFS: 61.25 chunks with stdev of 13.8
  - Fixed chunk size of 4 MB
- PeerStripe: 3.72 chunks with stdev of 3.1
  - Average chunk size 81.28 MB with stdev 19.9 MB

→ Fewer chunks in PeerStripe allows

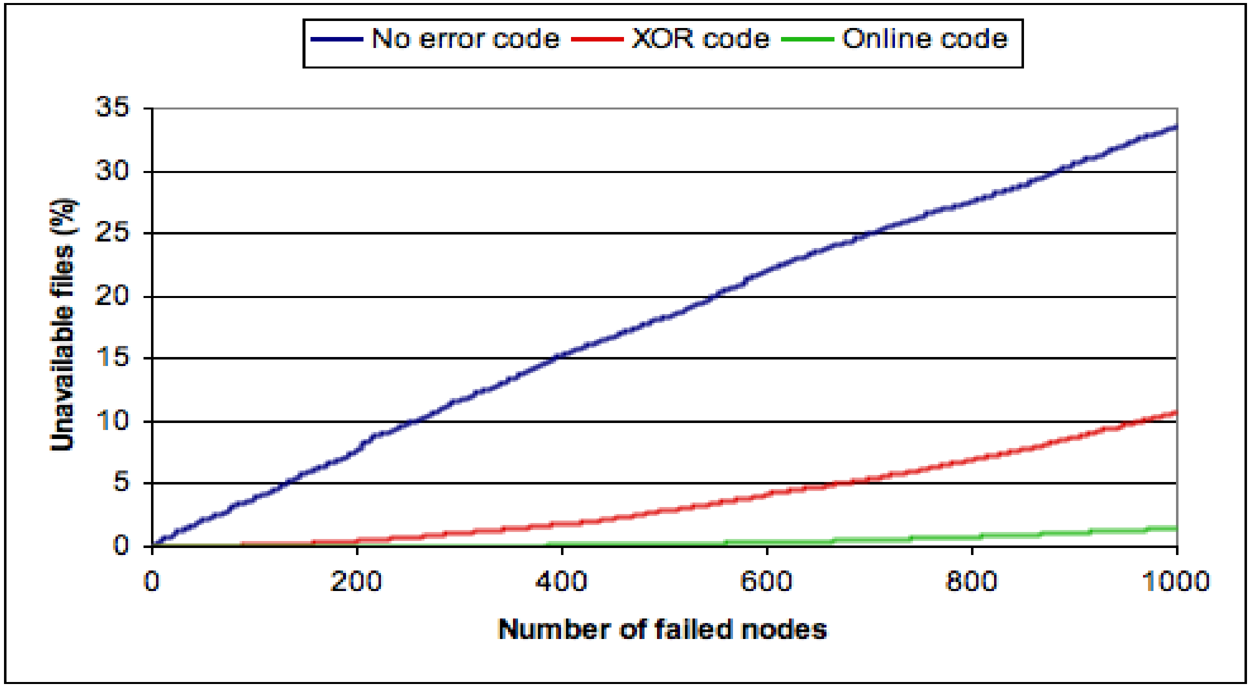
- Fewer expensive p2p lookups
- Performance similar to PAST

Overall System Capacity Utilization



- PeerStripe: 20.19% better than PAST
- PeerStripe: 7.18% better than CFS
- PeerStripe can utilize the available storage capacity more efficiently even at higher utilization

Error Coding: File Availability



- XOR code - 23% less failures
- Online code - 32% less failures
- Online code provides excellent fault tolerance against node failures

VirginiaTech Invent the Future

Error Coding Performance

- Compare XOR (1:1) and Online code with NULL code

Erasure code	Encoded size		Encoding time	
	Size(MB)	Overhead	Time	Overhead
Null	4	0%	11	0%
XOR	6	50%	79	618%
Online	4.12	3%	264	2300%

- XOR - factor of 3.3 times faster than online codes
- Online code - slower than XOR,
  - Decoding can start as soon as a block becomes available and can be overlapped with retrieval of other blocks
- The efficiency of online code overshadows its overhead

Effects of Participant Churn

- Failed up to 20% of total nodes

Nodes failed (percentage of total)	Data lost	Data regenerated		
	Total (GB)	Total (GB)	Average (GB)	Sd (GB)
10 percent	0	28044.35	28.04	79.85
20 percent	142.18	58625.78	29.31	80.02

- 29.3 GB of data was regenerated per node failure
- Total of 58,625.8 GB regenerated
- 142.2 GB data was lost which is small compared to the 278.7 TB of total data
- The data recreated per failure is small: 0.01%

Verification on PlanetLab

- 40 different distributed sites
- Number of failed stores reduced by
  330% w.r.t. PAST
  105% w.r.t. CFS
- Storage utilization:
  CFS 52%, PAST - 47%, PeerStripe - 63%
- Online codes provided **98.6%** availability through four node failures

VirginiaTech
Invent the Future

Interfacing with Condor

- Utilize a 32-node Condor pool
- CFS and PeerStripe worked for smaller files
- DHT lookups introduced an overhead - few for PeerStripe
- Overhead for PeerStripe is small

QuickTime™ and a
decompressor
are needed to see this picture.

Outline

- Begin to our Means
- End to our Means
- Experimental Study
- Conclusion

Conclusion

- P2p-based storage can be extended with erasure coding and striping to provide robust, scalable, and reliable distributed storage for scientific computing.

- PeerStripe achieves better utilization of collective capacity of nodes with good performance
- Error coding is effective in providing fault tolerance and data availability
- Multicast can be used for replica maintenance
- Use of interposing library allows easy integration with new and existing applications

Questions?

- chmille3@cs.vt.edu
- butta@cs.vt.edu

- http://research.cs.vt.edu/dssl/