Towards a Parallel High-Performance Search Engine
Tobias Berka and Marian Vajteršic
University of Salzburg, Austria

**Content Management**
- Add documents for indexing,
- Remove documents,
- Change documents,
- Track users, groups and permissions.

**Additional Functionality**
- Store and track categories or tags,
- Handle versions and multi-part texts,
- Multi-linguality is a big issue.

**Parallel Document Storage**
- Store raw plain-text documents,
- Straight-forward storage,
- Simplify or eliminate mark-up,
- Convert external IDs to internal surrogates,
- Keep track of access rights.

**Key Aspects**
- Resilient storage and recovery,
- Text analysis.

**Statistical Dictionary**
- Compute document frequency,
- Select indexing terms,
- Prepare weights for term weighting,
- Identify multi-word expressions (MWEs).

**Tasks and Algorithms**
- Compute counts and frequencies,
- Compute significance of MWEs,
- Test decomposability of MWEs.

**Exact Search**
- Retrieval on a very sparse index,
- Search with very few key terms,
- Use pre-ranked lists per key term,
- Parallelism: balanced queries on replicated indices.

**Important Caveat: Sparseness Breaks Easily**
- Accounting for synonyms breaks sparseness,
- Query-by-example breaks sparseness,
- Interactive query refinement breaks sparseness.

**Parallel Sparse Index**
- Collect document vectors in a matrix,
- Very sparse; less than 1% non-zeroes,
- Store documents by term for retrieval,
- Store terms by document for analysis,
- Access rights still apply.

**Parallelization Issues**
- Balanced storage versus balanced workload,
- Different tasks with different requirements.

**Vector Construction**
- Trim stop-words ("the", "and", "or", etc.),
- Map words to word-stems ("going" to "go", etc.),
- Match and count indexing terms and MWEs,
- Apply term weighting (TF-IDF),
- Normalize vector length.

**Advanced Issues**
- Resilient matching,
- Deep parsing.

**Accurate Search**
- Partition features and documents,
- Obtains super-linear speed-up with MPI,
- Small parallel components are most effective,
- Replication to increase throughput.

**Further Optimizations**
- Cluster-based and multi-cluster search,
- Result cursors and caching,
- Pipelined query processing.

**Parallel Dense Index**
- Persistently and reliably store dense vectors,
- Efficient index replication needed,
- Dynamic system scaling required (nodes, disks),
- High reliability needed.

**Many Open Issues**
- Application-specific fault tolerance is essential,
- Highly efficient distributed middleware required,
- Effective parallel/distributed storage needed.

**Dimensionality Reduction**
- Improve the sparse vector representation,
- Account for synonymy, polysemy, etc.,
- From long, sparse vectors to short, dense vectors,

**Our New Method**
- Rare term vector replacement,
- As good as latent semantic indexing (LSI),
- More stable w.r.t. choice of parameters,
- Patent free.