MapReduce for Machine Learning on Multicore

Cheng-Tao Chu et al.

Presented by Yi Zhang

Feb 24, 2009

With contents borrowed from:
http://www.cs.duke.edu/courses/spring08/cps399.28/lectures/02-google.pdf

Pattern Recognition and Machine Learning, Christopher M. Bishop

Why Multicore

• Gains from frequency scaling are diminishing
  – Memory Wall: disparity of CPU/memory speed
  – ILP Wall: difficult to find enough parallelism in single instruction stream via compilers etc.
  – Power Wall: leakage current, heat, ...
• Putting more processing cores on single chip!
  – Ideally would get linear speedup
  – Compared to multiprocessors: faster inter-core, on-chip communication

How to Utilize Multicore?

• Non-trivial to fully utilize multicore
  – Nature of applications?
  – Programming facilities?
  – Manual rewrite required for many applications
    • Reduces, prone to error: race conditions, deadlocks,...
• Question:
  – A general programming method on multicore?
    Particularly for machine learning?

One Approach

• The problem domain
  – Machine learning algorithms with certain properties
• The tool
  – Google’s MapReduce
• The goals
  – A general method for exploiting parallelism in many algorithms
  – Taking advantage of multicore

The Summation Form

• Many ML algorithms fit the Statistical Query Model
  – Learning is based on the global properties of large samples
  – Expressible in the form of summations over samples
  – Summation can be done independently, in parallel
    • Divide the data set into pieces
    • Assign each piece to a core
    • Aggregate the results at the end
• Can be expressed by Google’s MapReduce framework

MapReduce

• Large-scale data processing
  – Want to use 1000s of machines, but don’t want hassle of managing things
• MapReduce provides
  – Automatic parallelization & distribution
  – Fault tolerance
  – I/O scheduling
  – Monitoring & status updates
map / reduce

- Programming model from Lisp (and other functional languages)
  - \( (\text{map} \ \text{square} \ '(1 \ 2 \ 3 \ 4)) \Rightarrow (1 \ 4 \ 9 \ 16) \)
  - \( (\text{reduce} + \ '(1 \ 4 \ 9 \ 16)) \Rightarrow 30 \)
- Many problems can be phrased this way
- Easy to distribute
- Nice failure/retry semantics

The MapReduce Model

- \( \text{map}(\text{key}, \text{val}) \Rightarrow (\text{new-key}, \text{new-val}), \ldots \)
  - Run on each item in an input set
  - Emit a list of key-value pairs
- \( \text{reduce}(\text{key}, \text{vals}) \Rightarrow \text{output} \)
  - Run for each unique key and all associated values emitted by map()
  - Emit output

Example: Counting words in docs

- **Input**: \((\text{url}, \text{contents})\) pairs
- **map**(key=\text{url}, val=\text{contents}):
  - For each word \(w\) in \text{contents}, emit \((w, 1)\)
- **reduce**(key=\text{word}, vals=\text{counts}):
  - Sum up all values in \text{counts}
  - Emit result \((\text{word}, \text{sum})\)

MapReduce in Picture

**Example 1: Linear Regression**

- **Problem**
  - Given training set \((x_1, y_1), \ldots, (x_m, y_m)\), find estimate \(\theta^*\) in linear model \(y = \theta^T x\), s.t. is minimized
  - Equivalent formulation: \(y = \theta^T x + \epsilon\), where \(\epsilon \sim N(0, \beta)\); find the maximum likelihood estimate for \(\theta^*\)
- **Least square estimate**
  - \(d = \sum_{i=1}^{m} (x_i, x'_i)\) and \(b = \sum_{i=1}^{m} y_i x'_i\)
- **MapReduce**
  - Two sets of mappers divide the task of computing \(A\) and \(b\)
  - Two reducers sum up partial results for \(A\) and \(b\)
  - Algorithm finally computes \(A^{-1}b\)
- **Speedup**
  - Multicore: \(O(m n^2 + n^2 \log n + n^2 \log P)\) vs. single core: \(O(n^2 + n^2)\)
Example 2: Naïve Bayesian

- Problem
  - Given a set of feature/label pairs: \((x_1, y_1), \ldots, (x_m, y_m)\) and a new point \(x_j\), predict its label \(y_j\). Assume \(y_j=0\) or \(1\).
- Naïve Bayesian classifier
  - Pick \(y=y_j\) that maximizes \(p(y|x_j)\) for a new \(x_j\)
  \[
p(y|x_j) \propto P(y)P(x_j|y)
  \]
  - Assume independence among components of \(x\) given \(y\)
    \[
p(y|x_j) \propto P(y)P(x_{j,1}|y)P(x_{j,2}|y)\ldots P(x_{j,n}|y)
  \]
    - Need \(P(y=0), P(y=1), P(x_{j,i}=k|y=0), P(x_{j,i}=k|y=1)\)

MapReduce
- Partition training set \(\{1, \ldots, m\} = M_1 + M_2 + \ldots + M_P\) for \(P\) mappers
- Mappers compute \(\ldots\)
- Reducers combine partial results, e.g.,

\[
\sum_{i \in M_1} y_i = P_{11}, \sum_{i \in M_2} y_i = P_{10} \quad \sum_{i \in M_P} y_i = P_{00}
\]
- Speedup: \(O(n + c)\) vs. \(O(nm/P + nc \log P)\)

Example 3: k-means

- Distance computing step:
  - Splitting data into subgroups
  - Mapper computes the distances of each point to all the centroids and puts it in the appropriate cluster
- New centroids computing step:
  - Splitting data into subgroups
  - Mapper emits pairs \(<\text{clusterId}, \text{partial sums}>\>
  - Reducer aggregates partial sums according to cluster IDs and computes new centroids

Example 4: Principal Component Analysis

- PCA
  - Widely used for dimensionality reduction, lossy data compression, feature extraction, etc.
- Formulation (Maximum Variance)
  - Consider a set of observations \(x_1, \ldots, x_m\) with dim \(n\)
  - Project data onto a space with dim \(d < n\) s.t. the variance of the projected data is maximized
  - Assume \(d\) is given

PCA (2)
- Simple Case \(d=1\)
  - Suppose data is projected to a line represented by vector \(u\) in the \(n\)-dimensional space
  - Suppose \(u\) is a unit vector: \(u^Tu=1\) (because only direction matters)
  - Each data point \(x\) is projected to a scalar \(u^T x\)
  - Variance of projected data: \(\text{var}(u^T x) = \frac{1}{m} \sum((u^T x_i - \text{mean})^2) = u^T S u\)
  - Maximization of \(u^T S u + \lambda (1-u^Tu)\) gives
    - \(u^T S u = \sum_{i=1}^{d} \lambda_i \)
    - Variance maximized when \(u\) is equal to the eigenvector having the largest eigenvalue \(\lambda\) (noting \(\lambda = \text{Var}\)).
  - Additional principal components can be defined incrementally

PCA (3)
- To obtain the data covariance matrix
  \[
  S = \frac{1}{m-1} \sum_{i=1}^{m} (x_i - \mu)(x_i - \mu)^T
  \]
  - Already in summation form
  - Each mapper produces partial sums
  - Reducer sums up partial results and computes \(S\)

Results
Conclusion

- **Many ML algorithms can utilize multicores**
  - Inherent parallelism: summation form
  - General framework: MapReduce

- **Major contributions**
  - Case-by-case discussion of common ML algorithms
  - Extensive experiments

- **Discussion**
  - How many cores are we talking about?
  - Communication overhead, reduce bottleneck, ...
  - MapReduce may be an overkill?
    - Mappers/reducers work on <key,val> pairs, but here just matrices etc. to be summed
    - Not using the failure handling techniques in MapReduce
  - Many other algorithms are not in summation form
    - e.g. Stochastic gradient ascent