# ConversationsBetweenDougMarvinAndGrant

The conversation that started it all:

```
Hi Doug,

I am interested in taking on #11 at
http://wiki.apache.org/jakarta-lucene/Lucene2Whiteboard and was wondering
if you had some time to elaborate on your ideas.

I would also like to be able to store more information at higher levels,
such as the document and the index.

If you would like, we can move this discussion to the dev list, but I
thought we could have a few conversations first and then I could work up
some interfaces and documentation over the next month or so to present
to the other developers.

Thanks,
Grant
```

Follow ups:

Grant Ingersoll wrote:
> I am interested in taking on #11 at
> http://wiki.apache.org/jakarta-lucene/Lucene2Whiteboard and was
> wondering if you had some time to elaborate on your ideas.

Great!

I have not spent enough time thinking about it to have very elaborate ideas! One group to collaborate with is the folks who've recently been porting Lucene to other languages, like Marvin Humphrey (Perl) and David Balmain (Ruby), cc'd. I had originally imagined a java-centric approach for extensibility (e.g., storing class names in the index). Now I think that would be a mistake.

We probably should not try to support both arbitrary extensibility and cross-language portability in a single file format. Instead we should add a few flags to control what's stored. We don't want too many, or things get messy. So the art will be deciding which will make the biggest difference. Per-position boosts is one extreme, and boolean-only is the other. We should support those and existing options. Anything more should be carefully justified.

> I would also like to be able to store more information at higher levels,
> such as the document and the index.

That sounds reasonable. Per-index properties would be very convenient.

> If you would like, we can move this discussion to the dev list, but I
> thought we could have a few conversations first and then I could work up
> some interfaces and documentation over the next month or so to present
> to the other developers.

Or you could just start drafting something in public on the wiki, Lucene3 or somesuch. That way others can follow along. You might first focus on the features to be added, then on the file formats, then the new APIs and finally on back-compatibility. The point is we want to be able to someday drop back-compatibility and have the new design not have many remnants of the old. So first we should specify how we want things to look, then figure out how to get there back-compatibly. Does that make sense?

Cheers,

Doug

Hello Doug, hello Grant, thanks for bringing me in...

On May 8, 2006, at 10:25 AM, Doug Cutting wrote:

> Grant Ingersoll wrote:
>> I am interested in taking on #11 at http://wiki.apache.org/jakarta-
>> lucene/Lucene2Whiteboard and was wondering if you had some time to
>> elaborate on your ideas.
>
> We probably should not try to support both arbitrary extensibility
> and cross-language portability in a single file format. Instead we
> should add a few flags to control what's stored. We don't want too
> many, or things get messy. So the art will be deciding which will
> make the biggest difference. Per-position boosts is one extreme,
> and boolean-only is the other. We should support those and
> existing options. Anything more should be carefully justified.

I agree with all of the above. In addition...

I've been playing around with a rich position scheme using KinoSearch, and got it to work mechanically without changing any search logic, but I decided I wasn't going to get very far in the absence of a sophisticated search-time benchmarker. Writing one of those is job one. I'll probably get around to that eventually, but I'm not sure just when.

It would be interesting to see how much better a boolean scorer which implements the proximity algorithm described in the Google paper performs in terms of precision. This could also be done using the present index format. Has it been tried?

I originally thought that a fixed-width/variable-width hybrid for the rich positions scheme would yield better performance while preserving full proximity data rather than truncating as the Googs propose. However, thinking about the amount of calculation that will occur per position in a boolean scorer, my gut feeling is that the decompression overhead of a VInt wouldn't be a major factor. Again, benchmark to confirm.

As discussed on java-dev, integrating freqs, positions, and boosts/ norms into one contiguous section of one file ought to improve the ability of a Searcher to launch from rest and return a query. That's not terribly important in a persistent Java server setup, but it would be handy for say, a help file system, or for quick 'n' dirty Perl CGI.

The number of bits per position dedicated to weighting (4-8 out of 16) in the Google paper is maddeningly small. However the number of bits per document per term for a common term is embarrassingly large compared to the 8 Lucene currently has available. It strikes me that it might be helpful to delta encode not just positions, but boosts as well.

Marvin Humphrey
Rectangular Research
http://www.rectangular.com/

Marvin Humphrey wrote:
> The number of bits per position dedicated to weighting (4-8 out of  16)
> in the Google paper is maddeningly small.  However the number of  bits
> per document per term for a common term is embarrassingly large
> compared to the 8 Lucene currently has available.  It strikes me that
> it might be helpful to delta encode not just positions, but boosts as
> well.

Not sure what you mean here, since boosts are not ordered.

Personally I think the eight-bit floats used by Lucene give plenty of
precision for this class of computation.  Relevant documents should be
easily distinguished from non-relevant documents, and fine-differences
in ranking between relevant documents don't matter.  The only time folks
have complained about the precision of eight-bit floats in Lucene is
when they've attempted to overload them with other semantics besides
relevance (e.g., dates), which is inappropriate.

So I would opt for delta-encoded positions with a one-byte boost.

I think combining frequencies and positions in a single file might be
useful.  If folks don't want to pay the penalty of pawing through
positions then they should disable position indexing for that field.

Another thing folks have frequently asked for is per-document boosts
(i.e., boosts instead of frequencies, and no positions).

Some useful posting options are thus:

a. <doc>+
b. <doc, boost>+
c. <doc, freq, <position>+ >+
d. <doc, freq, <position, boost>+ >+

These suggest the following booleans per field:

1. freq
2. document boost
3. position (requires freq)
4. position boost (requires position)

Doug

On May 9, 2006, at 9:07 AM, Doug Cutting wrote:

> Marvin Humphrey wrote:
>> The number of bits per position dedicated to weighting (4-8 out
>> of  16) in the Google paper is maddeningly small.  However the
>> number of  bits per document per term for a common term is
>> embarrassingly large  compared to the 8 Lucene currently has
>> available.  It strikes me that  it might be helpful to delta
>> encode not just positions, but boosts as  well.
>
> Not sure what you mean here, since boosts are not ordered.

Within a termdoc[1], each position will probably have the same boost
by default.  Since Lucene doesn't currently implement boost-per-
position, that's effectively what we have now.

It seemed wasteful to allocate one byte to each position for boost
information when all of them are the same.  A scheme similar to what
is currently used in the .frq file would potentially save us some
space: left shift the position delta by 1 and use the low bit to
indicate whether or not there has been a change in the boost since

the last position.

I originally thought it might make sense to delta encode the boost if
there was a change.  But I also wasn't necessarily assuming that the
scoring multiplier associated with each position would be an 8-bit
float.  I haven't come up with a better idea yet, though.  :)

> Personally I think the eight-bit floats used by Lucene give plenty
> of precision for this class of computation.

I agree, I like the 8 bit floats.  But going down even to 7 bits
radically reduces the range that can be covered by that float.  7
might still be enough, but 4 I think is too few.  1/16 of the
quantization levels, but fully half of the space requirement --
feh.   Reminds me of the difference between 16-bit audio and 8-bit
audio.  Ergo, my objection to that aspect of the Google98 scheme.

What does it mean to have the 8-bit float available per position?

Right now, TermScorer gets the norm/boost information by grabbing a
byte from the a norms array, then decoding it with a lookup in the
normDecoder.  Under boost-per-position, we'd have to ask for the
boost by calling termPositions.getBoost().  I think responsibility
would fall to a SegmentTermPositions object to iterate through the
positions and build up that number.

Say we opt for simple summing.  That means that at index time, we
calculate a total boost for the termdoc, and then divvy it up per
position in slices proportional to each position's existing boost
(which would be 1 by default).  I'm afraid that means that our search-
time sum would have a compounded quantization error and we'd be worse
off than we are now.

> I think combining frequencies and positions in a single file might
> be useful.  If folks don't want to pay the penalty of pawing
> through positions then they should disable position indexing for
> that field.

While I suspect that merging frequencies and positions is justified
if only because it cuts down on disk seeks, I'm not sure that
disabling position indexing is a realistic option for one very common
use case: a bodytext field that you need to be able to perform phrase
queries against.

> Some useful posting options are thus:
>
> a. <doc>+
> b. <doc, boost>+
> c. <doc, freq, <position>+ >+
> d. <doc, freq, <position, boost>+ >+
>
> These suggest the following booleans per field:
>
> 1. freq
> 2. document boost
> 3. position (requires freq)
> 4. position boost (requires position)

Delicious!

IMO this discussion should be taking place on java-dev.  I grant
permission to forward my contributions in full if we all agree and
somebody wants to post the thread there in one chunk.

Marvin Humphrey
Rectangular Research
http://www.rectangular.com/

[1] I've taken to using "termdoc" to mean "one term indexing one
document" and "posting" to mean "one term indexing one document one

time".

Marvin Humphrey wrote:
> Within a termdoc[1], each position will probably have the same boost by
> default. Since Lucene doesn't currently implement boost-per- position,
> that's effectively what we have now.
>
> It seemed wasteful to allocate one byte to each position for boost
> information when all of them are the same. A scheme similar to what is
> currently used in the .frq file would potentially save us some space:
> left shift the position delta by 1 and use the low bit to indicate
> whether or not there has been a change in the boost since the last
> position.

Right. That makes sense. But when boosts change they could go up or
down, and by large amounts, so it's not clear that we should write the
difference rather than simply the new value.

> What does it mean to have the 8-bit float available per position?
>
> Right now, TermScorer gets the norm/boost information by grabbing a
> byte from the a norms array, then decoding it with a lookup in the
> normDecoder. Under boost-per-position, we'd have to ask for the boost
> by calling termPositions.getBoost(). I think responsibility would fall
> to a SegmentTermPositions object to iterate through the positions and
> build up that number.
>
> Say we opt for simple summing. That means that at index time, we
> calculate a total boost for the termdoc, and then divvy it up per
> position in slices proportional to each position's existing boost
> (which would be 1 by default). I'm afraid that means that our search-
> time sum would have a compounded quantization error and we'd be worse
> off than we are now.

I'm confused. If we're sticking the norm into the position boost, then
we just multiply it into every position boost before they're summed,
rather than multiplying them into the sum as we do now, right? So we
don't have to divide up the norm. Am I missing something?

My thinking is that one could specify both position boosts and norms, in
which case both would be multiplied into the score. Or one could omit
norms and instead store their values in position boosts, or vice versa.

For a phrase, we might boost it by the average of it's matching term's
boosts.

>> I think combining frequencies and positions in a single file might be
>> useful. If folks don't want to pay the penalty of pawing through
>> positions then they should disable position indexing for that field.
>
> While I suspect that merging frequencies and positions is justified if
> only because it cuts down on disk seeks, I'm not sure that disabling
> position indexing is a realistic option for one very common use case: a
> bodytext field that you need to be able to perform phrase queries against.

Right. Disabling positions should be an option on a field-by-field basis.

>> Some useful posting options are thus:
>>
>> a. <doc>+
>> b. <doc, boost>+
>> c. <doc, freq, <position>+ >+
>> d. <doc, freq, <position, boost>+ >+
>>
>> These suggest the following booleans per field:
>>
>> 1. freq

>> 2. document boost
>> 3. position (requires freq)
>> 4. position boost (requires position)
>
>
> Delicious!
>
> IMO this discussion should be taking place on java-dev.  I grant
> permission to forward my contributions in full if we all agree and
> somebody wants to post the thread there in one chunk.

If I have a chance, I'll do that.  Better yet, we should post a summary
to the wiki as a design proposal.

Doug

On May 9, 2006, at 12:42 PM, Doug Cutting wrote:

> If we're sticking the norm into the position boost, then we just
> multiply it into every position boost before they're summed, rather
> than multiplying them into the sum as we do now, right?  So we
> don't have to divide up the norm.  Am I missing something?

No, it was my mistake.  I've worked through the problem and found my
error.  I'm down with 8-bit floats for the position boosts.

Marvin Humphrey
Rectangular Research
http://www.rectangular.com/

At the Index (Directory ???) level, I think we could make it such that a
Directory/Index can have Fields (perhaps just Keyword based, not sure
what it would mean to have tokenized text stored on the index)
associated with it as well.  Thus we would be able to use existing
mechanisms for writing Index level metadata.  We would just need a new
place in the file format for storing these fields.  This may call for
some marker interfaces (in Java land anyway) such that the Index field
addition mechanism only allows certain kind of Fields if that makes sense.

Then the user could do similar things to an Index that they do to a
Document (i.e. get value for a field, etc.).  I currently simulate this
in our IR Tools implementation by writing out an XML file that goes in
the same directory as the index files and stores metadata about the index.

Doug Cutting wrote:
> Marvin Humphrey wrote:
>> The number of bits per position dedicated to weighting (4-8 out of
>> 16) in the Google paper is maddeningly small.  However the number of
>> bits per document per term for a common term is embarrassingly large
>> compared to the 8 Lucene currently has available.  It strikes me
>> that  it might be helpful to delta encode not just positions, but
>> boosts as  well.
>
> Not sure what you mean here, since boosts are not ordered.
>
> Personally I think the eight-bit floats used by Lucene give plenty of
> precision for this class of computation.  Relevant documents should be
> easily distinguished from non-relevant documents, and fine-differences
> in ranking between relevant documents don't matter.  The only time
> folks have complained about the precision of eight-bit floats in
> Lucene is when they've attempted to overload them with other semantics
> besides relevance (e.g., dates), which is inappropriate.
>
> So I would opt for delta-encoded positions with a one-byte boost.

> I think combining frequencies and positions in a single file might be
> useful.  If folks don't want to pay the penalty of pawing through
> positions then they should disable position indexing for that field.
>
> Another thing folks have frequently asked for is per-document boosts
> (i.e., boosts instead of frequencies, and no positions).
>
> Some useful posting options are thus:
>
> a. <doc>+
> b. <doc, boost>+
> c. <doc, freq, <position>+ >+
> d. <doc, freq, <position, boost>+ >+
>
> These suggest the following booleans per field:
>
> 1. freq
> 2. document boost
> 3. position (requires freq)
> 4. position boost (requires position)
>
> Doug
>

--

Grant Ingersoll
Sr. Software Engineer
Center for Natural Language Processing
Syracuse University
School of Information Studies
335 Hinds Hall
Syracuse, NY 13244

http://www.cnlp.org
Voice:  315-443-5484
Fax: 315-443-6886

I would also be interested in optionally storing more information about
the Term in the term dictionary (such as POS or other metadata).   But
this also might be more added to the Term Vector capabilities instead
and thus wouldn't cause overhead during indexing/search


Doug Cutting wrote:
> Marvin Humphrey wrote:
>> The number of bits per position dedicated to weighting (4-8 out of
>> 16) in the Google paper is maddeningly small.  However the number of
>> bits per document per term for a common term is embarrassingly large
>> compared to the 8 Lucene currently has available.  It strikes me
>> that  it might be helpful to delta encode not just positions, but
>> boosts as  well.
>
> Not sure what you mean here, since boosts are not ordered.
>
> Personally I think the eight-bit floats used by Lucene give plenty of
> precision for this class of computation.  Relevant documents should be
> easily distinguished from non-relevant documents, and fine-differences
> in ranking between relevant documents don't matter.  The only time
> folks have complained about the precision of eight-bit floats in
> Lucene is when they've attempted to overload them with other semantics
> besides relevance (e.g., dates), which is inappropriate.
>
> So I would opt for delta-encoded positions with a one-byte boost.
>
> I think combining frequencies and positions in a single file might be
> useful.  If folks don't want to pay the penalty of pawing through
> positions then they should disable position indexing for that field.
>
> Another thing folks have frequently asked for is per-document boosts
> (i.e., boosts instead of frequencies, and no positions).
>
> Some useful posting options are thus:
>
> a. <doc>+
> b. <doc, boost>+
> c. <doc, freq, <position>+ >+
> d. <doc, freq, <position, boost>+ >+
>
> These suggest the following booleans per field:
>
> 1. freq
> 2. document boost
> 3. position (requires freq)
> 4. position boost (requires position)
>
> Doug
>

--

Grant Ingersoll
Sr. Software Engineer
Center for Natural Language Processing
Syracuse University
School of Information Studies
335 Hinds Hall
Syracuse, NY 13244

I think storing a String->String map per index makes sense, but I'm not
sure it makes sense for this to be implemented as a set of Fields, since
many field attributes do not make sense here, such as Indexed,

Tokenized, Stored, Vectored, etc.  The Binary & Compressed attributes
could make sense however.  So we could try to refactor Field, but much
of it would not be shared code anyway, since we would probably not use
the same format for field names.  So I think I would opt for a similar
yet distinct implementation for index attributes rather than overloading
Field.

Doug

Grant Ingersoll wrote:
> At the Index (Directory ???) level, I think we could make it such that a
> Directory/Index can have Fields (perhaps just Keyword based, not sure
> what it would mean to have tokenized text stored on the index)
> associated with it as well.  Thus we would be able to use existing
> mechanisms for writing Index level metadata.  We would just need a new
> place in the file format for storing these fields.  This may call for
> some marker interfaces (in Java land anyway) such that the Index field
> addition mechanism only allows certain kind of Fields if that makes sense.
> Then the user could do similar things to an Index that they do to a
> Document (i.e. get value for a field, etc.).  I currently simulate this
> in our IR Tools implementation by writing out an XML file that goes in
> the same directory as the index files and stores metadata about the index.
>
> Doug Cutting wrote:
>
>> Marvin Humphrey wrote:
>>
>>> The number of bits per position dedicated to weighting (4-8 out of
>>> 16) in the Google paper is maddeningly small.  However the number of
>>> bits per document per term for a common term is embarrassingly large
>>> compared to the 8 Lucene currently has available.  It strikes me
>>> that  it might be helpful to delta encode not just positions, but
>>> boosts as  well.
>>
>>
>> Not sure what you mean here, since boosts are not ordered.
>>
>> Personally I think the eight-bit floats used by Lucene give plenty of
>> precision for this class of computation.  Relevant documents should be
>> easily distinguished from non-relevant documents, and fine-differences
>> in ranking between relevant documents don't matter.  The only time
>> folks have complained about the precision of eight-bit floats in
>> Lucene is when they've attempted to overload them with other semantics
>> besides relevance (e.g., dates), which is inappropriate.
>>
>> So I would opt for delta-encoded positions with a one-byte boost.
>>
>> I think combining frequencies and positions in a single file might be
>> useful.  If folks don't want to pay the penalty of pawing through
>> positions then they should disable position indexing for that field.
>>
>> Another thing folks have frequently asked for is per-document boosts
>> (i.e., boosts instead of frequencies, and no positions).
>>
>> Some useful posting options are thus:
>>
>> a. <doc>+
>> b. <doc, boost>+
>> c. <doc, freq, <position>+ >+
>> d. <doc, freq, <position, boost>+ >+
>>
>> These suggest the following booleans per field:
>>
>> 1. freq
>> 2. document boost
>> 3. position (requires freq)
>> 4. position boost (requires position)
>>
>> Doug
>>

>

On May 10, 2006, at 7:31 AM, Grant Ingersoll wrote:

> I would also be interested in optionally storing more information
> about the Term in the term dictionary (such as POS or other
> metadata).  But this also might be more added to the Term Vector
> capabilities instead and thus wouldn't cause overhead during
> indexing/search

Scanning through the term dictionary is, if I am not mistaken, a pretty resource-intensive task. However, the thing about Term Vectors is they're document-centric. What are some of the uses you envision?

What do you mean by POS? The Wikipedia disambiguation page yields some entertaining possibilities: <http://en.wikipedia.org/wiki/POS>. I'm guessing part of speech?

Marvin Humphrey
Rectangular Research
http://www.rectangular.com/

You can, but this then requires more sophisticated querying techniques, right? Such as PrefixQuery or WildcardQuery when you don't care about those things.

I am sure it would have perf. implications. It would have to be implemented such that when it is turned off there is negligible perf. impact.

Doug Cutting wrote:
> Grant Ingersoll wrote:
>> I would also be interested in optionally storing more information
>> about the Term in the term dictionary (such as POS or other metadata).
>
> Can't you simply pack this onto the end of the term string?
> Adding per-term attribute sets could have performance implications.
>
> Doug

--

Grant Ingersoll
Sr. Software Engineer
Center for Natural Language Processing
Syracuse University
School of Information Studies
335 Hinds Hall
Syracuse, NY 13244

On May 10, 2006, at 9:54 AM, Grant Ingersoll wrote:

> Yep, Part of Speech.  CNLP is a pseudo academic research lab
> (meaning we do commercial work too), we often have rich information
> about terms, phrases, etc. from our NLP analysis, such as
> categories, co-references, etc.  One of our biggest usages of
> Lucene is in our QA system, where these kind of things come in
> handy.  I agree, though, that this is probably not worth the
> performance hit unless there is a way to make it negligible when it
> is turned off.  In our QA system, the performance hit I think I
> would see in Lucene is most likely small potatoes compared to
> having to lookup and process some of these things later.  I can
> envision some nice SpanQuery uses that let me identify candidate
> answers very quickly as compared to having to post-process them
> from the reconstructed document.  At any rate, there is probably a
> very small audience for this, so it may not be worth it.

This seems like the kind of app where a Lucene/RDBMS hybrid could
serve.  Adding arbitrary metadata is an SQL-like feature.  I imagine
you've explored that possibility, though.  :)

Maybe you could do some hacking with a synonym filter that stored
part of speech as a Term at the same position.  Is that what you're
doing with the TokenFilter?

> Another one of the things we need to from time to time is calculate
> corpus statistics, i.e. the total number of occurrences for a term
> (usually for all terms).  This involves looping over all the terms
> and the term docs and summing and storing (perhaps I am missing
> something in the API?).

No, you're not missing anything.  You have to use a TermDocs object
and sum termDocs.freq() over all docs for each term.  The Term
Dictionary only gives you docFreq.

How often does the index get updated?

> This can be done using a TokenFilter, but then we have to manage
> the memory, data structures, etc. whereas, Lucene could easily
> store this count along with the term.  This kind of thing could
> also be stored at the index level, and is part of what I imagined
> for that functionality.

It's also easy to export.  But exporting's a problem if you require
frequent updates.

Marvin Humphrey
Rectangular Research
http://www.rectangular.com/