ElasticSearch vs MongoDB

NoSQL means different things to different people and/or databases. They should be compared based on what properties you need (and can sacrifice), and not by what "category" of NoSQL-things they fit into.

Here's an article I wrote on the subject: Elasticsearch as a NoSQL database

It is certainly possible to use Elasticsearch as a primary store, when the limitations described are not showstoppers. One good example is when using Logstash. Logstash is a fantastic tool for managing logs and shoving them into Elasticsearch, perhaps also archiving them somewhere else just in case. Logs are write once, read many. No updating, no need for transactions, integrity constraints, etc.

What about systems like Postgres, that come with full-text search *and* ACID-transactions? (Other examples are the full-text capabilities of MySQL, MongoDB, Riak, etc.) While you can implement basic search with Postgres, there's a huge gap both in possible performance, and in the features. As mentioned in the section on transactions, Elasticsearch can "cheat" and do a lot of caching, with no concern for multi version concurrency control and other complicating things. Search is also more than finding a keyword in a piece of text: it's about applying domain specific knowledge to implement good relevancy models, giving an overview of the entire result space, and doing things like spell checking and autocompletion. All while being *fast*.

Elasticsearch is commonly used *in addition* to another database. A database system with stronger focus on constraints, correctness and robustness, and on being readily and transactionally updatable, has the master record - which is then asynchronously pushed to Elasticsearch. (Or pulled, if you use one of Elasticsearch's "rivers".)

I typically use PostgreSQL and ZooKeeper as keeper of truths, which we feed into Elasticsearch for awesome searching.

Like with everything else, there's no silver bullet, no one database to rule them all. That's likely to always be the case, so know the strengths and weaknesses of your stores!

NoSQL databases are storages in more traditional sense, and the difference here is a balance between various features of the storage. ElasticSearch is good for specific task - indexing and searching big datasets. It is used when you have some secondary info about your data and you need to know actual records to select. And since it's architecture is optimized for this, it's weaker in some other use cases. For example, in compare to many NoSQL databases ElasticSearch is slow on adding new data.

Another important thing about ElasticSearch and similar tools is that indexing semantics is defined on client side, so the actual indexing cannot be optimized as well as with real storages. At the same time, this provides an abstraction from various search APIs used by NoSQL databases, so underlying storage is not integrated into app too deep.

In practice, ElasticSearch (as like as Solr or AWS CloudSearch) is often used together with NoSQL and SQL databases, where database is used as persistent storage, and ElasticSearch is used for doing complex search queries, based on data content. For example, we at Thumbtack has succesfully used ElasticSearch for full-text indexing on top of MongoDB, which had no support for this before the latest version.

So the difference in responsibilities is obvious, but I think that with further development of NoSQL databases, especially such feature as server-side scripts and after accepting some standard on querying facilities (like SQL language), search functionality can be fully absorbed by storages.

alking about arguments *to use* Mongo instead of/together with ES:

1. User/role management.
    - Built-in in MongoDB. May not fit all your needs, may be clumsy somewhere, but it exists and it was implemented pretty long time ago.
    - The only thing for security in ES is `shield`. But it ships only for Gold/Platinum subscription for production use.
2. Schema
    - ES is schemaless, but its built on top of `Lucene` and written in `Java`. The core idea of this tool - index and search documents, and working this way requires index consistency. At back end, all documents should be fitted in flat `lucene` index, which requires some understanding about how ES should deal with your nested documents and values, and how you should organize your indexes to maintain balance between speed and data completeness/consistency. Working with ES requires you to keep some things about schema in mind constantly. I.e: as you can index almost anything to ES without putting corresponding mapping in advance, ES can "guess" mapping on the fly but sometimes do it wrong and sometimes implicit mapping is evil, because once it put, it can't be changed w/o reindexing whole index. So, its better to not treat ES as schemaless store, because you can step on a rake some time (and this will be *pain* :)), but rather treat it as schema-intensive, at least when you work with documents, that can be sliced to concrete fields.
    - Mongo, on the other hand, can "chew and leave no crumbs" out of almost anything you put in it. And most your queries will work fine, `til you remember how Mongo will deal with your data from JavaScript perspective. And as JS is weakly typed, you can work with really schemaless workflow (for sure, if you need such)
3. Handling non-table-like data.

- ES is limited to handle data without putting it to search index. And this solution is good enough, when you need to store and retrieve some extra data (comparing to data you want to search against).
- MongoDB supports `gridFS`. This gives you ability to handle large chunks of data behind the same interface. I.e., you can store binary data in Mongo and retrieve it within the same interface, from your code perspective.

In simple words, elastic search used to index the data in MongoDB that you think need be indexed.